



بنيان الحواسيب (1)

الجزء العملي





منشورات جامعة دمشق
كلية الهندسة المعلوماتية

بنيان الحواسيب (1) الجزء العملي

المهندسة

عبير محمد ديب ميّاً

مشرفة على الأعمال في قسم النظم والشبكات الحاسوبية

1439-1440هـ

2017-2018م

جامعة دمشق



الفهرس

7	مقدمة الكتاب
11	الفصل الأول أداء المعالجات CPU Performance
33	الفصل الثاني تمثيل الأعداد في الحاسوب
45	الفصل الثالث الحساب في الحاسوب
61	الفصل الرابع البرمجة بلغة التجميع MIPS
105	الفصل الخامس ممر المعطيات Data Path
129	الفصل السادس توسيع الذاكر Memory Expansion
145	ملحق دليل التعامل مع برنامج محاكاة معالج MIPS
187	مسرد المصطلحات
195	المراجع



بسم الله الرحمن الرحيم

مقدمة الكتاب

أضع بين أيدي طلابنا الأعزاء كتاباً يغطي الجزء العملي لمادة بنيان الحواسيب (1) ليسير جنباً إلى جنب مع المنهاج النظري الذي يُدرّس في هذه المادة، وعلى أمل أن يدعم الأفكار والمعلومات التي تعلمها الطالب نظرياً بالاطلاع والتدرب على كمّ وافر من الأمثلة المحولة والتمارين التي تُركت بدون حل في نهاية الفصول بحيث يختبر الطالب نفسه في استيعاب ما قد مرّ معه خلال دراسته لكل فصل على حدة.

وكان من دواعي سروري أن أقدم لطلابنا كتاباً باللغة العربية وفاءً مني لهذه اللغة العظيمة وتأكيداً على غناها وكفاءتها، خاصة أن سورية كان لها السبق في اعتمادها اللغة العربية في التعليم الجامعي حتى في الفروع العلمية التي يتوقع صعوبة تدريسها باللغة العربية أو صعوبة تعريب مصطلحاتها، وقد حرصت في سياق الكتاب على استخدام المصطلحات العربية حتى يعتادها الطالب، مع ذكر المصطلح الانكليزي المقابل للمصطلح العربي في أول ورود له، كما وضعت مسرداً لأهم المصطلحات العربية المستخدمة في الكتاب مع ما يقابلها باللغة الانكليزية في نهاية الكتاب.

تهدف مادة بنيان الحواسيب بشكل رئيسي إلى تعريف الطالب بالبنية الداخلية للنظم الحاسوبية وطرائق قياس أدائها، وأهم المعاملات المؤثرة في الأداء (دور المعالج، والذاكرة الرئيسية، والذاكرة الخابية Cache.. إلخ). كما تهدف إلى إكساب الطلاب المهارات اللازمة لبرمجة المعالجات الصغيرة بلغة التجميع Assembly. ويغطي هذا الكتاب الجانب العملي من المادة وذلك بتناول المواضيع السابقة في مسائل وتمارين عديدة.

يتألف هذا الكتاب من ستة فصول، بالإضافة إلى ملحق في نهايته.

يُعنى الفصل الأول بدراسة أداء المعالجات، ويعرض من خلال الأمثلة والتمارين أهم المؤشرات والمعايير للمفاضلة بين أداء المعالجات، ومدى تأثير هذه المؤشرات على تقييم أداء المعالجات.

أما الفصل الثاني فيشمل على أمثلة وتمارين في أنظمة العد مما يلزم الطالب خلال دراسته لمنهاج بنيان الحواسيب، بما فيها من كيفية التحويل بين أنظمة العد المختلفة، وتمثيل الأعداد الحقيقية، وتمثيل الأعداد السالبة، بالإضافة إلى تمثيل الأعداد ذات الفاصلة العائمة.

ويتناول الفصل الثالث أمثلة وتمارين عن العمليات الحسابية من جمع وطرح وضرب وقسمة على الأعداد الثنائية، ثم على الأعداد ذات الفاصلة العائمة.

ويركّز الفصل الرابع من خلال الأمثلة المحولة والتمارين المتنوعة على البرمجة بلغة التجميع Assembly، وكيفية التعامل مع الذاكرة، والمصفوفات، واستخدام الحلقات، والمكدس، بالإضافة إلى التعرف على بعض التعليمات الزائفة pseudo instruction وما يقابلها من تعليمات حقيقية، بالإضافة إلى التعرف على أنماط التعليمات في معالج MIPS، وتمارين عن التحويل من تعليمات لغة التجميع إلى تعليمات لغة الآلة، وبالعكس.

يتناول الفصل الخامس مسائل عن ممر المعطيات Data Path بنوعيه أحادي الدور ومتعدد الأدوار، وذلك من أجل مجموعة من تعليمات المعالج MIPS التي سبق دراستها في الفصل الرابع، بالإضافة إلى مسائل حول أداء ممرات المعطيات المختلفة.

أما الفصل السادس فيتناول مفهوم توسيع الذاكر، وذلك بتجميع عدد من رقاقات الذاكر للحصول على ذاكرة أكبر، إما بعرض الكلمة أو بعدد الكلمات، أو بالاثنتين معاً، وتغطي مسائل هذا الفصل طرق التوصيل اللازمة في أنواع التوسيع المذكورة آنفاً.

وفي نهاية الفصول أضفت ملحقاً هو عبارة عن دليل للتعامل مع أحد برامج محاكاة المعالج MIPS وهو برنامج MARS، وقد اخترت هذا البرنامج لما يتميز به من سهولة التعامل معه، وواجهته الرسومية التي تعرض محتوى الذاكرة والسجلات عند تنفيذ البرامج المكتوبة بلغة التجميع MIPS، ولسهولة كشف الأخطاء عند وجودها، ويحوي الملحق أيضاً مجموعة من الأمثلة المحلولة وهي عبارة عن برامج جاهزة للتنفيذ، ليتدرب الطالب من خلالها على البرمجة بلغة التجميع، وكيفية التعامل مع برنامج المحاكاة، وكيفية الاستفادة من مزاياه، وقمت في نهاية كل مثال محلول بعرض صورة لنتيجة تنفيذ البرنامج التي يفترض الحصول عليها، مع التركيز فيها على القيم التي تهتم الطالب في كل مثال ليتأكد بذلك من صحة عمله.

وفي الختام أرجو أن يحقق هذا الكتاب الفائدة المنشودة منه، والله ولي التوفيق.

م. عبير ميا



الفصل الأول

أداء المعالجات CPU Performance

مقدمة:

لابد للمقارنة بين أداء عدة معالجات من توفر مقاييس يمكن الاعتماد عليها للوصول إلى المفاضلة بين أداء تلك المعالجات، وقد اشتهر في هذا المجال مجموعة من المعايير والمؤشرات التي تؤثر بشكل أو بآخر على أداء المعالجات، وفي هذا الفصل سنتعرف على أهم هذه المؤشرات ومدى تأثيرها على تقييم أداء المعالجات.

تردد المعالج f (Frequency): تقاس سرعة الساعة Clock في الحاسوب بعدد الذبذبات التي تصدرها في الثانية وهو ما يسمى الهرتز Hertz، فهو إذاً عدد دورات الساعة التي يعمل عليها المعالج في الثانية، ويساوي مقلوب دور الساعة ($f=1/T$) ووحدة القياس له هي هرتز Hz.

عدد الدورات الوسطي في التعليمات (CPI cycle per instruction):

عدد دورات الساعة التي يحتاجها المعالج لتنفيذ التعليمات الواحدة، ويؤخذ عادةً الوسطي منه في حال عدم تساوي عدد الدورات اللازمة لتنفيذ كل نوع من أنواع التعليمات، فمثلاً تعليمات الضرب لها عادةً CPI أعلى من تعليمات الجمع، وتعليمات التخزين والتحميل من الذاكرة لها CPI أعلى من تعليمات الجمع بين محتوى السجلات، وكذلك الأمر بالنسبة لتعليمات الفاصلة العائمة وتعليمات الأعداد الصحيحة. فمن أجل برنامج معين إذا كان تنفيذ التعليمات من كل نوع يلزمه عدداً معيناً من دورات الساعة مختلفاً عن عدد دورات الساعة في باقي التعليمات، فإن عدد الدورات الوسطي للتعليمات CPI بالنسبة للبرنامج المعطى ينتج عن مجموع جداءات عدد الدورات اللازم لتنفيذ التعليمات من كل نوع من أنواع التعليمات CPI في نسبة تكرار هذا النوع Fi في البرنامج، أو عدد دورات تنفيذ كامل البرنامج CC (Cycle Count) مقسوماً على عدد تعليمات البرنامج المنفذ

N (أو يرمز لعدد التعليمات أحياناً IC)، أي أن:

$$CPI = \frac{\sum (CPI_i \times F_i)}{N} = \frac{CC}{N}$$

زمن التنفيذ أو زمن المعالج CPUtime: وهو الزمن اللازم للمعالج فقط لإتمام المهمة دون النظر إلى زمن الدخل والخرج والزمن المتعلق بنظام التشغيل وغيره. والتناسب عكسي بين الأداء وزمن التنفيذ، ويعتبر زمن التنفيذ أدق المعايير لقياس الأداء.

$$Performance = 1/CPUtime$$

ويحسب زمن التنفيذ بجداء عدد دورات الساعة للبرنامج كاملاً CC في زمن دورة الساعة T، أو بتقسيم هذا العدد على تردد الساعة f.

وبما أن عدد دورات الساعة للبرنامج = عدد التعليمات * عدد الدورات الوسطي للتعليمات: $CC = N * CPI$ وبالتالي فإن:

زمن التنفيذ = عدد التعليمات * عدد الدورات الوسطي للتعليمات * دور الساعة

$$CPUtime = N * CPI * T = N * CPI / f = CC / f$$

ومن العلاقة السابقة يمكن حساب القيمة CPI كما يلي:

$$CPI = CPUtime * f / N$$

مقياس MIPS (Million Instruction per second): وهو عدد

ملايين التعليمات التي يمكن تنفيذها في الثانية.

$$MIPS = \frac{N}{CPUtime} \times 10^{-6} = \frac{f}{CPI} \times 10^{-6}$$

مقياس Mops (Million Instruction Per Second): عدد ملايين

العمليات التي يستطيع المعالج القيام بها في الثانية.

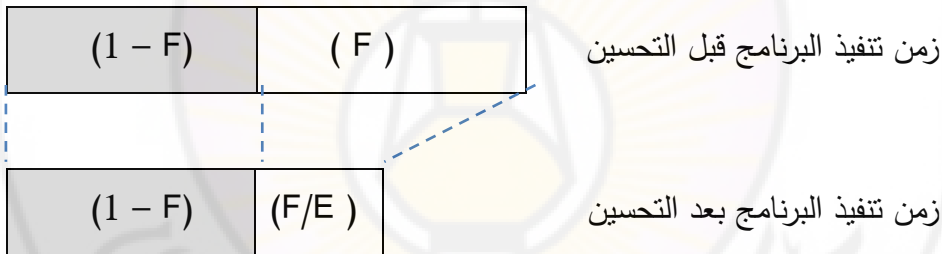
مقياس MFLOPS (Million Floating Point Operation Per)

(Second): عدد ملايين عمليات الفاصلة العائمة التي يستطيع المعالج القيام بها

في الثانية.

قانون أمدال Amdahl's Law: يوضح قانون أمدال مدى الربح في الأداء أو مقدار التسريع Speedup الذي يتوقع من حاسوب معين عند تحسين أحد معاملات أدائه. وبشكل عام، فإن الربح في الأداء أو التسريع يساوي مدة التنفيذ الكامل للمهمة قبل التحسين مقسوماً على مدة تنفيذ نفس المهمة بعد إجراء التحسين. فإذا كانت النسبة المئوية من زمن التنفيذ التي تتأثر بتطبيق التحسين هي F (Fraction)، وكان مقدار التحسين هو E (Enhancement)، بينما باقي زمن التنفيذ لا يتأثر بالتحسين ونسبته $(1-F)$ ، فإن:

$$\text{Speedup} = \frac{\text{Performance}(\text{new})}{\text{Performance}(\text{old})} = \frac{\text{CPUtime}(\text{old})}{\text{CPUtime}(\text{new})}$$



$$\text{CPUtime}_{\text{new}} = (1/E) * F * \text{CPUtime}(\text{old}) + (1-F) * \text{CPUtime}(\text{old})$$

$$\text{CPUtime}_{\text{new}} = ((F/E) + (1-F)) * \text{CPUtime}_{\text{old}}$$

$$\text{Speedup} = \frac{1}{\frac{F}{E} + (1-F)}$$

أي يمكن حساب مقدار التسريع الناتج عن تحسين معين إما عن طريق تحديد النسبة بين زمني التنفيذ قبل وبعد التحسين، أو باستخدام قانون أمدال بعد معرفة قيمة كل من النسبة التي أجري عليها التحسين F ، ومقدار التحسين E .

أمثلة محلولة:

المثال (1-1): بفرض أن أحد البرامج يستغرق تنفيذه على المعالج P1 ذي التردد 2GHz مدة 4 ثوان، وعند تنفيذه باستخدام المعالج P2 لوحظ أنه يحتاج عدد دورات ساعة أكثر بمرتين من حالة المعالج P1 ويستغرق تنفيذه مدة 5 ثوان فقط. المطلوب حساب تردد المعالج P2 الذي يحقق المواصفات المذكورة.

الحل:

$$\text{CPU time}(P1) = \frac{CC(P1)}{f(P1)}$$

$$4s = \frac{CC(P1)}{2 \times 10^9 \text{ cycles/s}} \Rightarrow CC(P1) = 8 \times 10^9 \text{ cycles}$$

$$\text{CPU time}(P2) = 5 = \frac{2 * CC(P1)}{f(P2)} \Rightarrow f(P2) = 3.2 \text{ GHz}$$

ماذا تلاحظ عند المقارنة بين سرعة تنفيذ البرنامج على كل من المعالين وتردد المعالين؟ وهل يمكن الاعتماد على تردد المعالجات كمؤشر وحيد عند المقارنة بين أداء المعالجات، أم لا بد من مراعاة باقي المعايير المؤثرة على زمن تنفيذ البرامج؟.

المثال (2-1): بفرض لدينا ثلاثة معالجات P1, P2, P3 تقوم بتنفيذ نفس البرنامج، ولها الترددات وعدد الدورات الوسطي بالتعليمات موضحة في الجدول التالي:

المعالج P _i	التردد f _i	عدد الدورات بالتعليمات CPI _i
P1	2 GHz	1.5
P2	1.5 GHz	1.0
P3	3 GHz	2.5

أ- أي المعالجات له الأداء الأفضل في تنفيذ هذا البرنامج؟

ب- بفرض أن المعالج P1 ينفذ برنامجاً معيناً في 10 ثانية، احسب عدد دورات الساعة وعدد التعليمات في هذه الحالة.

الحل:

أ- لنفرض أن البرنامج مؤلف من N تعليمة، فإن:

$$\text{CPU time}(P1) = 1.5 N / (2 \times 10^9) = 0.75 N \times 10^{-9}$$

$$\text{CPU time}(P2) = N / (1.5 \times 10^9) = 0.66 N \times 10^{-9}$$

$$\text{CPU time}(P3) = 2.5 N / (3 \times 10^9) = 0.83 N \times 10^{-9}$$

بما أن المعالج P2 له زمن التنفيذ الأقل فهو ذو الأداء الأفضل.

ب- حسب المعطيات في المعالج P1 فإن:

$$10 = 1.5 \times N_1 / (2 \times 10^9) \Rightarrow$$

$$N_1 \text{ (number of instructions)} = 1.33 \times 10^{10} \text{ instr.}$$

$$\text{CC1} = 1.5 \times N_1 = 1.5 \times 1.33 \times 10^{10} = 2 \times 10^{10} \text{ cycle}$$

المثال (3-1): يحاول أحد مصممي المترجمات compiler designer

المفاضلة بين تسلسلي تعليمات لبرنامج معين ينفذان على أحد المعالجات، علماً أنه يوجد ثلاثة أنماط مختلفة للتعليمات A, B, C، حيث يلزم لتنفيذ التعليمة من النمط A دورة واحدة، ومن النمط B دورتين، ومن النمط C ثلاثة أدوار.

فإذا علمت أن تسلسل التعليمات الأول مؤلف من 5 تعليمات: تعليمتين من النمط A وتعليمة من النمط B وتعليمتين من النمط C، بينما تسلسل التعليمات الثاني مؤلف من 6 تعليمات: 4 تعليمات من النمط A وتعليمة واحدة من النمط B وواحدة من النمط C.

المطلوب: أ- أيهما أسرع في التنفيذ؟

ب- احسب CPI في الحالتين.

الحل:

أ- لحساب زمن التنفيذ نحسب عدد أدوار الساعة في البرنامج:

$$CC1 = \sum(CPI_i \times N_i) = (2 \times 1) + (1 \times 2) + (2 \times 3) = 10 \text{ cycles}$$

$$CC2 = \sum(CPI_i \times N_i) = (4 \times 1) + (1 \times 2) + (1 \times 3) = 9 \text{ cycles}$$

وبما أن زمن دورة الساعة نفسه في الحالتين وبالتالي فزمن التنفيذ في الحالة الأولى أكبر من زمن التنفيذ في الحالة الثانية، فتكون سرعة التنفيذ في الحالة الثانية أعلى من الحالة الأولى.

ب- حساب عدد دورات الساعة الوسطي في التعليمة بالنسبة للبرنامج في

الحالتين المذكورتين:

$$CPI1 = 10/5 = 2$$

$$CPI2 = 9/6 = 1.5$$

نلاحظ أن CPI في الحالة الثانية أقل من الحالة الأولى، ولكن لا تكفي مقارنة CPI لإعطاء مؤشر على الأداء الأفضل، بل لا بد من مراعاة قيم المعاملات الثلاثة: تردد الساعة وعدد التعليمات بالإضافة إلى وسطي عدد الدورات في التعليمة CPI من أجل تحديد الأداء الأفضل.

المثال (4-1): تم تجربة كل من المترجمين compilerA، compilerB

لبرنامج معين prog فوجد أن زمن التنفيذ وعدد التعليمات في كل من الحالتين كما يلي:

$$\text{compilerA: } N = 1 \times 10^9 \text{ inst. , CPU time} = 1 \text{ s}$$

$$\text{compilerB: } N = 1.2 \times 10^9 \text{ inst. , CPU time} = 1.4 \text{ s}$$

احسب CPI في الحالتين علماً أن زمن دورة الساعة للمعالج $T = 1 \text{ ns}$.

الحل:

$$\text{CPU time} = \text{CPI} \times N \times T \Rightarrow \text{CPI} = \text{CPU time} / (N \times T)$$

$$\text{CPI}_A = 1 / (10^{-9} \times 10^9) = 1$$

$$\text{CPI}_B = 1.4 / (10^{-9} \times 1.2 \times 10^9) = 1.167$$

المثال (5-1): ليكن لدينا معالج يعمل بتردد 500MHz، وفيه: كل تعليمة حسابية أو منطقية ALU_Instruction يلزمها 3 دورات ساعة، وكل تعليمة تفرع أو قفز branch/jump يلزمها دورتي ساعة، بينما تعليمة التخزين في الذاكرة sw يلزمها 4 دورات ساعة، وتعليمة التحميل من الذاكرة lw يلزمها 5 دورات ساعة. وبفرض أن برنامجاً ما ينفذ مجموع التعليمات التالية:

x=200 million ALU instructions,

y=55 million branch/jump instructions,

z=25 million sw instructions,

w=20 million lw instructions.

المطلوب: حساب زمن التنفيذ CPUtime .

الحل:

الطريقة الأولى: حساب عدد دورات الساعة لكامل البرنامج:

$$\text{CC} = (x \times 3 + y \times 2 + z \times 4 + w \times 5) = 910 \times 10^6 \text{ clock cycles}$$

$$\text{CPUtime} = \text{CC} / f = 910 \times 10^6 / (500 \times 10^6) = 1.82 \text{ sec}$$

الطريقة الثانية: حساب عدد الدورات الوسطي في التعليمة CPI للبرنامج

المعطى:

$$\text{CPI} = \text{CC} / N$$

$$\text{CPI} = (x \times 3 + y \times 2 + z \times 4 + w \times 5) / (x + y + z + w) = 3.03$$

$$\text{CPU time} = N \times \text{CPI} / f$$

$$= (x+y+z+w) \times 3.03 / (500 \times 10^6)$$

$$= 300 \times 10^6 \times 3.03 / (500 \times 10^6) = 1.82 \text{ sec}$$

المثال (1-6): يبين الجدول التالي النسبة المئوية لكل نمط من أنماط التعليمات المنفذة في أحد البرامج، وعدد دورات الساعة في كل منها:

نوع التعليمات	عدد أدوار الساعة بالتعليمات	نسبة التكرار
Branch	2	15%
Store	4	5%
Load	5	30%
ALU	4	50%

المطلوب: حساب عدد دورات الساعة الوسطي في البرنامج.

الحل:

$$\text{CPI} = 0.5 \times 4 + 0.3 \times 5 + 0.05 \times 4 + 0.15 \times 2$$

$$\text{CPI} = 4 \text{ cycles/instruction}$$

المثال (1-7): يراد اختبار مترجمين مختلفين على معالج يعمل بتردد 4 GHz، ويفرض وجود ثلاثة أنماط مختلفة للتعليمات A,B,C تتطلب على التوالي 1,2,3 دورة ساعة. تم استخدام كل من المترجمين من أجل برنامج معين، فحصلنا على مزيج التعليمات التالية:

عدد التعليمات مقدرة بالمليار	النمط A	النمط B	النمط C
------------------------------	---------	---------	---------

1	1	5	باستخدام المترجم الأول
1	1	10	باستخدام المترجم الثاني

المطلوب: أ- أي التسلسلين sequence أسرع حسب زمن التنفيذ؟

ب- أي التسلسلين الناتجين له قيمة MIPS أعلى؟ ماذا تستنتج؟

الحل: أ- حساب زمن التنفيذ:

$$\text{CPUtime} = \text{CC} / f$$

$$\text{CC1} = ((5 \times 1) + (1 \times 2) + (1 \times 3)) \times 10^9 = 10 \times 10^9 \text{ cycle}$$

$$\text{CC2} = ((10 \times 1) + (1 \times 2) + (1 \times 3)) \times 10^9 = 15 \times 10^9 \text{ cycle}$$

$$\text{CPUtime1} = 2.5 \text{ seconds}$$

$$\text{CPUtime2} = 3.75 \text{ seconds}$$

ب- حساب MIPS : حسب القانون فإن:

$$\text{MIPS} = N / \text{CPUtime} \times 10^6$$

$$\text{MIPS1} = 2800$$

$$\text{MIPS2} = 3200$$

بالمقارنة نجد أن: $\text{CPUtime2} > \text{CPUtime1}$, $\text{MIPS2} > \text{MIPS1}$

أي أن المعيار MIPS أحياناً قد يتناسب عكساً مع الأداء.

المثال (1-8): نريد المقارنة بين أداء جهازين مختلفين M1, M2، فتم إجراء

القياسات التالية على كل منهما:

البرنامج	الزمن على M1	الزمن على M2
الأول	10 ثوان	5 ثوان
الثاني	3 ثوان	4 ثوان

وإذا علمت أن عدد تعليمات البرنامج الأول المنفذة على M1 يساوي 200 مليون
تعليمية، وعلى M2 يساوي 160 مليون تعليمية، المطلوب:

- أ- أي الجهازين أسرع بالنسبة لكل من البرنامجين مع تحديد النسبة؟
 ب- احسب قيمة MIPS للبرنامج الأول عند تنفيذه على كل من M1, M2.
 ج- إذا علمت أن تردد الساعة في M1 هو 2GHz وفي M2 هو 3GHz
 احسب CPI لكل من M1, M2 من أجل البرنامج الأول.
 د- بفرض أن CPI للبرنامج الثاني على كل من M1, M2 هو نفسه للبرنامج
 الأول عند تنفيذه على كل من M1, M2، احسب عدد التعليمات N للبرنامج الثاني
 المنفذ على كل من M1, M2 اعتماداً على القيم المعطاة في الطلبات السابقة.
 الحل:

- أ- من أجل البرنامج الأول فإن M2 أسرع بنسبة $2=10/5$ مرة من M1.
 ومن أجل البرنامج الثاني فإن M1 أسرع بنسبة $1.33=4/3$ مرة من M2.
 ب- حساب قيمة MIPS للحالتين:

$$MIPS1 = N1 / (CPUtime1 * 10^6) = (200 * 10^6) / (10 * 10^6) = 20$$

$$MIPS2 = N2 / (CPUtime2 * 10^6) = (160 * 10^6) / (5 * 10^6) = 32$$

- ج- حساب قيمة CPI للحالتين من القانون $MIPS = (f / CPI) * 10^{-6}$ وبالتالي

فإن:

$$CPI = (f / MIPS) * 10^{-6}$$

$$CPI1 = (2 * 10^9 / 20) * 10^{-6} = 100$$

$$CPI2 = (3 * 10^9 / 32) * 10^{-6} = 94$$

د- نحسب عدد التعليمات من قانون زمن تنفيذ البرنامج الثاني على كل من M1, M2 كما يلي:

$$CPUtime = N * CPI / f$$

$$N1 = CPUtime1 * f1 / CPI1 = 3 * 2 * 10^9 / 10 = 600 * 10^6 \text{ inst.}$$

$$N2 = CPUtime2 * f2 / CPI2 = 4 * 3 * 10^9 / 9.4 = 1277 * 10^6 \text{ inst.}$$

المثال (1-9): بفرض وجود جهازين متشابهين لكن لهما بنية مجموعة تعليمات ISA مختلفة، أحدهما يدعم تعليمات الأعداد الحقيقية MFP، والآخر لا يدعم تعليمات الأعداد الحقيقية MNFP، وكل من الجهازين يعمل على التردد 1000MHz، الجهاز ذو العتاد الداعم للعمليات على الأعداد الحقيقية مباشرة MFP له المواصفات التالية:

كل تعليمة ضرب للأعداد الحقيقية يلزمها 6 دورات ساعة.

كل تعليمة جمع للأعداد الحقيقية يلزمها 4 دورات ساعة.

كل تعليمة قسمة للأعداد الحقيقية يلزمها 20 دورة ساعة.

أي تعليمة أخرى بما فيها التعليمات على الأعداد الصحيحة يلزمها 2 دورة.

أما الجهاز الآخر MNFP فلا يملك عتاداً خاصاً للفاصلة العائمة، وبالتالي لا يدعم تعليمات الأعداد الحقيقية السابقة بشكل مباشر، وإنما من أجل كل تعليمة على الأعداد الحقيقية يقوم المترجم بإضافة تسلسل من تعليمات الأعداد الصحيحة يكافئ عمل تعليمة الأعداد الحقيقية:

كل تعليمة ضرب للأعداد الحقيقية يلزمها 30 تعليمة أعداد صحيحة.

كل تعليمة جمع للأعداد الحقيقية يلزمها 20 تعليمة أعداد صحيحة.

كل تعليمة قسمة للأعداد الحقيقية يلزمها 50 تعليمة أعداد صحيحة.

وكل تعليمة أعداد صحيحة في MNFP يلزمها 2 دورة ساعة.
وبفرض أن البرنامج P فيه مزيج التعليمات التالية: 10% ضرب للأعداد الحقيقية، 15% جمع للأعداد الحقيقية، 5% قسمة للأعداد الحقيقية، 70% باقي التعليمات.

المطلوب:

- أ- احسب قيمة MIPS في كل من الجهازين.
- ب- بفرض أن البرنامج P على الجهاز MFP مؤلف من 300,000,000 تعليمة، احسب زمن تنفيذ البرنامج على كل من الجهازين.
- ج- احسب قيمة MFLOPS لكل من الجهازين.

الحل:

أ- لحساب MIPS نقوم بحساب CPI أولاً:

$$CPI_{MFP} = 0.1 \times 6 + 0.15 \times 4 + 0.05 \times 20 + 0.7 \times 2 = 3.6$$

$$CPI_{MNFP} = 2$$

$$MIPS_{MFP} = f / CPI * 10^6 = 270.3$$

$$MIPS_{MNFP} = 500$$

بالمقارنة نجد أن : $MIPS_{MNFP} > MIPS_{MFP}$.

ب- نوجد عدد التعليمات في الجهاز MNFP حسب نسب التعليمات المعطاة في الجهاز MFP كما هو موضح في الجدول التالي:

عدد التعليمات (MFP)	عدد التعليمات (MNFP)	
$10^6 * 30$	$10^6 * 900$	تعليمات الضرب الحقيقية
$10^6 * 45$	$10^6 * 900$	تعليمات الجمع الحقيقية
$10^6 * 15$	$10^6 * 750$	تعليمات القسمة الحقيقية
$10^6 * 210$	$10^6 * 210$	باقي التعليمات
$10^6 * 300$	$10^6 * 2760$	المجموع

وبتطبيق قانون حساب زمن التنفيذ نجد:

$$\text{CPUtime (MFP)} = 300 \times 10^6 \times 3.6 / 1000 \times 10^6 = 1.08 \text{ s}$$

$$\text{CPUtime (MNFP)} = 2760 \times 10^6 \times 2 / 1000 \times 10^6 = 5.52 \text{ s}$$

ج- حساب MFLOP :

$$\text{Nfp_operations} = (30 + 45 + 15) \times 10^6 = 90 \times 10^6 \text{ FPOperation}$$

$$\text{MFLOPS} = \text{Nfp_operations} / \text{CPUtime} \times 10^6$$

$$\text{MFLOPS}_{\text{MFP}} = 90 \times 10^6 / 1.08 \times 10^6 = 83.3$$

$$\text{MFLOPS}_{\text{MNFP}} = 90 \times 10^6 / 5.52 \times 10^6 = 16.3$$

المثال (10-1): يراد إجراء تحسين على مخدم انترنت بحيث أن المعالج الجديد أسرع 10 مرات من المعالج القديم، فإذا علمت أن المعالج الأصلي يقضي 40% من وقته في المعالجة، و60% من الوقت في انتظار الدخل/الخرج. المطلوب حساب التسريع speedup الناتج عن هذا التحسين.

الحل:

النسبة التي يتم التحسين عليها من المعالج هي 40%، ومقدار التحسين 10

مرات، فيكون التسريع الحاصل بتطبيق قانون أمدال Amdahl's Law:

$$\text{Speedup} = \frac{1}{\frac{F}{E} + (1-F)}$$

$$\text{Speedup} = 1 / [0.4/10 + (1 - 0.4)] = 1.56$$

المثال (11-1): بفرض أن برنامجاً معيناً يلزمه 100 ثانية للتنفيذ على أحد الأجهزة، منها 80 ثانية لتنفيذ عمليات الضرب، المطلوب:

أ- ما هي نسبة التحسين على عمليات الضرب اللازمة لجعل البرنامج أسرع أربعة أضعاف؟

ب- ما هي نسبة التحسين على عمليات الضرب اللازمة لجعل البرنامج أسرع خمسة أضعاف؟

الحل:

أ- التسريع المطلوب speedup يساوي 4، فيمكن استنتاج مقدار التحسين E من العلاقة:

$$\text{Speedup} = \frac{1}{\frac{F}{E} + (1-F)}$$

$$\text{Speedup} = 4 = \frac{1}{\frac{0.8}{E} + (1-0.8)}$$

$$4 = E / [0.80 + (0.2)*E]$$

$$3.2 + 0.8 * E = E$$

$$0.2 * E = 3.2 \Rightarrow E = 16$$

أي أنه يجب تسريع إجراء الضرب بنسبة 16 مرة للحصول على تسريع في الأداء مقداره 4.

ب- التسريع المطلوب speedup هو 5، وبتطبيق قانون أمدال:

$$\text{Speedup} = \frac{1}{\frac{F}{E} + (1-F)}$$

$$5 = 1 / (0.8 / E + 0.2)$$

$$1 = 5 * 0.8 / E + 5 * 0.2$$

$$1 = 4 / E + 1 \Rightarrow 0 = 4 / E \Rightarrow E = 4 / 0 = \infty$$

أي أنه يستحيل تحقيق تسريع مقداره 5.

المثال (12-1): بفرض أنه من أجل برنامج قياس أداء benchmark معين تم اقتراح ثلاثة تحسينات ممكنة كما يلي:

$$E_1 = 30, F_1 = 15\%$$

$$E_2 = 20, F_2 = 15\%$$

$$E_3 = 10, F_3 = 70\%$$

المطلوب تحديد التحسين الذي يؤدي إلى أداء أفضل من بين التحسينات الثلاثة المقترحة.

الحل:

نحسب قيمة التسريع في كل من الحالات الثلاثة:

$$\text{Speedup}_{E_1} = \frac{1}{\frac{F_1}{E_1} + (1-F_1)} = \frac{1}{\frac{0.15}{30} + (1-0.15)} = \frac{1}{0.005 + 0.85}$$

$$\text{Speedup}_{E_1} = 1.169$$

$$\text{Speedup}_{E_2} = \frac{1}{\frac{F_2}{E_2} + (1-F_2)} = 1.166$$

$$\text{Speedup}_{E_3} = \frac{1}{\frac{F_3}{E_3} + (1-F_3)} = 2.703$$

بما أن التسريع الناتج عن التحسين الثالث Speedup_{E_3} هو الأعلى لذا يتم اختياره من أجل أداء أفضل.

المثال (1-13): بفرض لدينا برنامج مؤلف من 820,000,000 تعليمة، والتعليمات مقسمة إلى أربعة أنماط مختلفة بالشكل الموضح في الجدول التالي:

نمط التعليمة	CPIi (كما في معالج MIPS)	عدد التعليمات
Branch	3	150,000,000
Store	4	185,000,000
Load	5	260,000,000
ALU / R-type	4	225,000,000

أ- إذا كان زمن تنفيذ هذا البرنامج يساوي 1.57 s، احسب زمن دور الساعة الذي ينفذ عليه.

ب- بفرض أن 25% من مجموع تعليمات التحميل Load في البرنامج السابق يتبعها مباشرة تعليمات حسابية تستخدم المعطيات التي تم تحميلها للتو من الذاكرة، ولتسريع تنفيذ البرنامج تم اقتراح إضافة نمط جديد من التعليمات بحيث يصبح خرج ذاكرة المعطيات كأحد مداخل ALU، وبحيث أن هذه التعليمة الجديدة تحل مكان التعليمتين السابقتين المتتاليتين، ويلزم لهذه التعليمة 7 دورات ساعة.

المطلوب حساب التسريع الناتج عن هذا المقترح، بفرض أن تردد المعالج لم يتغير، وأن الإجابة على هذا الطلب مستقلة عن إجابة الطلب السابق (أ).

الحل:

أ- حسب قانون زمن التنفيذ فإن:

$$\text{CPUtime} = \text{CC} \times T$$

$$\Rightarrow 1.57s = [(3 \times 150,000,000) + (4 \times 185,000,000) + (5 \times 260,000,000) + (4 \times 225,000,000)] \times T$$

$$260,000,000) + (4 \times 225,000,000)] \times T$$

$$\Rightarrow 1.75s = (450 + 740 + 1300 + 900) \times 10^6 \times T = 3,390 \times 10^6 \times T$$

$$\Rightarrow T = 4.63 \times 10^{-10} \text{ s}$$

وبالتالي فإن تردد هذا المعالج $f=2.16 \text{ GHz}$

ب- نقوم أولاً بحساب عدد تعليمات كافة الأنماط بعد أن تمت إضافة النمط

الجديد:

- عدد تعليمات التفرع Branch والتخزين Store لم يتغير .

- عدد تعليمات التحميل Load: $260,000,000 \times 0.75 = 195,000,000$

- عدد تعليمات النمط الجديد ALU-Load:

$$260,000,000 \times 0.25 = 65,000,000$$

- عدد تعليمات ALU:

$$225,000,000 - 65,000,000 = 160,000,000$$

- عدد التعليمات الكلي = $755,000,000$ تعليمة.

وبالتالي يصبح زمن التنفيذ كالتالي:

$$\text{CPUtime} = \text{CC} \times T$$

$$= (3 \times 150,000,000 + 4 \times 185,000,000 + 5 \times$$

$$195,000,000 + 4 \times 160,000,000 + 7 \times 65,000,000) \times T$$

$$= 3,260,000,000 T$$

وبالمقارنة مع زمن التنفيذ الناتج في الطلب (أ) وهو $3,390,000,000T$ نجد:

$$\text{Speedup} = \frac{\text{CPUtime}(\text{old})}{\text{CPUtime}(\text{new})} = \frac{3,390,000,000 T}{3,260,000,000 T} = 1.04$$

أي أن التسريع الحاصل حوالي 4%.

تمارين غير محلولة

التمرين (1-1): بفرض أن لدينا برنامجاً معيناً مؤلفاً من التعليمات التالية:

نمط التعليمات	حسابية	تخزين	تحميل	تفرع
عدد التعليمات	500	50	100	50
عدد الدورات بالتعليمات	1	5	5	2

فما هو زمن تنفيذ هذا البرنامج على معالج ذي تردد 2 GHz؟

التمرين (2-1): بفرض لدينا تطبيق مكتوب بلغة البرمجة جافا، زمن تنفيذه على معالج لحاسوب محمول يساوي 15 s، وعند استخدام مترجم جافا جديد وُجد أنه يحتاج إلى 0.6 من عدد التعليمات اللازمة في المترجم القديم، ولكن قيمة CPI تزيد بنسبة 1.1.

المطلوب حساب زمن تنفيذ التطبيق باستخدام المترجم الجديد.

التمرين (3-1): الجدول التالي يظهر القياسات التي أجريت من أجل برنامج

معين على حاسوبين A , B:

Computer B	Computer A	
8 billion	10 billion	عدد التعليمات
4 GHz	4 GHz	تردد الساعة
1.0	1.1	CPI

أ- أي الحاسوبين له MIPS أعلى؟

ب- أيهما هو الأسرع؟

ملاحظة: 1 billion = 1000 millions = 1,000,000,000

التمرين (1-4): بفرض لدينا الجهازين M1 يعمل على التردد 1000MHz و M2 يعمل على التردد 1500MHz، ولدينا 4 أنماط مختلفة للتعليمات A,B,C,D لكل منها عدد دورات بالتعليمية كما هو موضح في الجدول التالي:

النمط	CPI (M1)	CPI (M2)
A	1	2
B	2	2
C	3	4
D	4	4

أ- إذا علمت أن عدد التعليمات المنفذة في أحد البرامج مقسم بالتساوي بين أنماط التعليمات السابقة، أي الجهازين أسرع في تنفيذ البرنامج وما هي النسبة بين الزمنين؟

ب- احسب تردد الجهاز M1 حتى يكون له نفس أداء الجهاز M2 ذي التردد 1500MHz.

التمرين (1-5): بفرض أنه يتم تنفيذ برنامج معين على جهاز، وفيه: العدد الكلي للتعليمات 10,000,000 تعليمة، CPI الوسطي = 2.5، $f = 2 \text{ GHz}$.
المطلوب:

أ- حساب زمن تنفيذ البرنامج.

ب- بتنفيذ نفس البرنامج مع استخدام مترجم آخر نتج عنه عدد تعليمات يساوي 9,500,000 تعليمة، $\text{CPI} = 3$ ، على معالج يعمل بتردد 3 GHz. ما هي نسبة

التسريع الناتجة عن هذه التغييرات.

التمرين (1-6): بفرض أن الجدول التالي يبين نسب التعليمات في برنامج معين، وقيم CPI لكل نوع من أنواع التعليمات:

نمط التعليمية	نسبة تكرار التعليمية	CPI
load/store	0.30	2
integer ALU	0.40	1
FP op	0.10	6
Branch	0.20	2

المطلوب:

أ- ما هي قيمة CPI الوسطي؟

ب- بفرض أن البرنامج السابق مؤلف من 100 مليون تعليمية، وأن تردد المعالج الذي تنفذ عليه هو 2GHz، ما هو زمن التنفيذ CPUtime لهذا البرنامج؟

ج- احسب قيمة MIPS.

د- احسب قيمة MFLOPS.

التمرين (1-7): بفرض أنه يتم تنفيذ برنامج مؤلف من التعليمات التالية:

التعليمات	نسبة تكرار التعليمات	عدد الدورات بالتعليمية
الحسابية والمنطقية ALU	50%	1
التحميل load	20%	5

1	%10	التخزين store
2	%20	التفرع branch

المطلوب: أي من التعديلات التالية سيؤدي لتحسين الأداء بشكل أفضل:

أ- اعتماد تقنية التنبؤ بالتفرع بحيث يقلل عدد الدورات في تعليمة التفرع إلى 1 دورة فقط.

ب- استخدام خابية معطيات Data Cache أفضل لتقليل عدد دورات تعليمة التحميل من الذاكرة إلى 3 دورات فقط.

التمرين (1-8): يبين الجدول التالي النسب المئوية لتكرار أنماط التعليمات

المختلفة في أحد البرامج، وعدد الدورات اللازمة لكل تعليمة منها:

نمط التعليمات	CPI(i)	نسبة تكرار التعليمات (Fi)
ALU	1	50%
Load	5	20%
Store	3	10%
Branch	2	20%

المطلوب:

أ- حساب CPI الوسطي.

ب- بفرض تم إجراء تحسين على تصميم المعالج بحيث تم خفض CPI

لتعليمات التحميل load من 5 إلى 2. ما هي نسبة التسريع الناتجة عن هذا التحسين.



الفصل الثاني

تمثيل الأعداد في الحاسوب

مقدمة:

تتطلب المكونات الحاسوبية وجود نظام عددي يتلاءم مع طبيعة تغذية الحاسوب بالطاقة الكهربائية وهذا ما يتحقق من خلال النظام الثنائي Binary system، وأساسه هو العدد 2، والجدول التالي يظهر القيم العشرية Decimal المقابلة لقوى العدد 2 لاستخدامها عند التحويل من النظام الثنائي إلى النظام العشري:

...	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	...
...	32	16	8	4	2	1	1/2	1/4	1/8	1/16	1/32	...

أما تحويل العدد من النظام العشري إلى الثنائي فيتم بالتقسيم المتكرر للعدد العشري على 2 مع الاحتفاظ بالباقي والذي يشكل بترتيبه قيمة العدد ثنائياً، إلى أن نحصل على ناتج القسمة مساوياً للصفر.

وفي حال وجود قسم كسري في العدد فيتم تحويل الأعداد الكسرية العشرية إلى ثنائية عن طريق الضرب المتكرر للعدد العشري بالأساس 2، وبعد كل عملية ضرب يتم الاحتفاظ بالقيمة الصحيحة الناتجة عن عملية الضرب والتي تساوي الصفر أو الواحد، وتكرر عملية الضرب للباقي بالقيمة 2 إلى أن تصبح قيمة الكسر مساوية الصفر، أو حتى نحصل على الدقة المطلوبة. وعند ذلك ترتب القيم الصحيحة المحتفظ بها بحيث أن أول رقم صحيح ناتج يتوضع على يمين الفاصلة مباشرة.

ولتسهيل التعامل مع الأرقام الثنائية وجد النظام الثماني octal حيث يمثل كل ثلاثة أرقام ثنائية برمز واحد من النظام الثماني، والنظام السداسي

عشر hexadecimal الذي يمثل فيه كل أربعة أرقام ثنائية برمز واحد من النظام السداسي عشر الذي يبدأ من العدد 0 إلى العدد F.

تمثيل العدد السالب ثنائياً: في العمليات الرياضية العادية يسمى العدد سالِباً إذا سبقته إشارة الناقص (-)، ويسمى موجباً إذا سبقته إشارة الزائد (+)، أما في الحاسوب فتستعمل ثلاث طرق لتمثيل الأعداد السالبة وهي:

طريقة التمثيل بالمطال والإشارة Sign-Magnitude Representation: حيث تحجز الخانة الأكثر أهمية (الخانة العليا من اليسار) للدلالة على إشارة العدد، واصطُـلح على استعمال الرقم 0 في العدد الموجب، و 1 في العدد السالب، أما الخانات المتبقية فتتمثل مطال العدد أي قيمته المطلقة.

طريقة الإتمام إلى واحد/ المتمم الأحادي One's Complement: وفيه يتم إيجاد متمم العدد وذلك بقلب الأصفار وحادات والعكس.

طريقة الإتمام إلى اثنين/ المتمم الثنائي Two's Complement: وفيه يتم إيجاد متمم العدد الممثل ثنائياً إلى واحد، ثم يجمع له العدد 1.

الأعداد ذات الفاصلة العائمة Floating Point Numbers: ويعبر عنها كعدد (جزء عشري) مضروب بثابت (الأساس) مرفوع إلى قوة ما (أس). ويستخدم هذا الترميز لتمثيل الأعداد الحقيقية الكبيرة والصغيرة جداً.

ويمكن تمثيل أعداد الفاصلة العائمة بثلاثة حقول: بت الإشارة (s) Sign، وحقول الأس (e) Exponent، وحقول الجزء العشري (f) Fraction. وتكتب بالشكل: $(\pm 1.f * 2^{\text{exponent}})$. ونلاحظ عدم الحاجة لتخزين الأساس 2، والعدد 1 على يسار الفاصلة.

s	e	f
---	---	---

وتستخدم معظم المعالجات المعيار IEEE754 لتمثيل أعداد الفاصلة العائمة، بالدقة المفردة single precision، أو الدقة المضاعفة double precision.

1- الدقة المفردة على 32 بت: $s=1 \text{ bit}, e=8 \text{ bit}, f=23 \text{ bit}$

2- الدقة المضاعفة على 64 بت: $s=1 \text{ bit}, e=11 \text{ bit}, f=52 \text{ bit}$

حيث يتم تخزين الأس المنحاز Biased exponent بعد جمع قيمة ثابتة تسمى الانحياز Bias إلى الأس الفعلي p للحصول على الأس المنحاز الممثل ثنائياً exp ، ويكون مقدار الانحياز 127 في الدقة المفردة، و 1023 في الدقة المضاعفة.

الجزء الكسري f	الأس المنحاز e	بت الإشارة s
دقة مفردة: 23	8	1
دقة مضاعفة: 52	11	1

القيم المحجوزة reserved bitpattern حسب المعيار IEEE754:

- الصفر (0) : ويمثل بالدقة المفردة بأصفار على 32 بت.
- $\pm\infty$: عندما تكون قيمة الأس المنحاز $e=255$ والجزء الكسري $f=0$.
- عدم التعيين NaN : عندما تكون قيمة الأس المنحاز $e=255$ والجزء الكسري $f \neq 0$.

أمثلة محلولة:

المثال (1-2): حوّل العدد الثنائي $(110101)_2$ إلى النظام العشري.

الحل:

$$\begin{aligned} 110101 &= 1*2^0 + 0*2^1 + 1*2^2 + 0*2^3 + 1*2^4 + 1*2^5 \\ &= 53 \end{aligned}$$

المثال (2-2): حوّل العدد الثنائي $(0.10011)_2$ إلى النظام العشري.

الحل:

$$\begin{aligned} 0.10011 &= 1*2^{-1} + 0*2^{-2} + 0*2^{-3} + 1*2^{-4} + 1*2^{-5} \\ &= 1/2 + 0 + 0 + 1/16 + 1/32 = 19/32 \\ &= 0.59375 \end{aligned}$$

المثال (3-2): حوّل العدد الثنائي $(110.101)_2$ إلى النظام العشري.

الحل:

$$\begin{aligned} 110.101 &= 0*2^0 + 1*2^1 + 1*2^2 + 1*2^{-1} + 0*2^{-2} + 1*2^{-3} \\ &= 2 + 4 + 1/2 + 1/8 \\ &= 6,625 \end{aligned}$$

المثال (4-2): ما هو تمثيل العدد $(98)_{10}$ في النظام الثنائي؟

الحل:

يتم الحصول على التمثيل الثنائي للعدد العشري بالتقسيم المتكرر للعدد المعطى على الأساس 2، والاحتفاظ بباقي القسمة، بحيث أن أول باقي ينتج عن القسمة يشكل البت الأدنى للمقابل الثنائي، وبالتالي فإن التمثيل الثنائي للعدد 98 هو:

2	(98) ₁₀ = (1100010) ₂
98	0
49	1
24	0
12	0
6	0
3	1
1	1
0	

كما يمكن الحصول عليه بجمع قوى العدد 2 الأصغر من العدد 98:

$$(98)_{10} = 2^6 + 2^5 + 2^1 = 64 + 32 + 2$$

$$(98)_{10} = (1100010)_2$$

المثال (2-5): حوّل الرقم العشري $(0.375)_{10}$ إلى النظام الثنائي.
الحل:

يمكن الحل ذهنياً حسب مجموع القوى السالبة الأصغر من العدد 0.375 أي:

$$(0.375)_{10} = 0 \cdot 0.5 + 1 \cdot 0.25 + 1 \cdot 0.125 = (0.011)_2$$

أو حسب الطريقة العامة:

$$0.375 \cdot 2 = 0.75 \Rightarrow "0"$$

$$0.75 \cdot 2 = 1.5 \Rightarrow "1"$$

$$0.5 \cdot 2 = 1.0 \Rightarrow "1"$$

$$\Rightarrow (0.375)_{10} = (0.011)_2$$

المثال (2-6): حول العدد $(11010101)_2$ إلى النظام العشري.
الحل:

$$\begin{aligned} (11010101)_2 &= 1 \times 2^0 + 0 \times 2^1 + 1 \times 2^2 + 0 \times 2^3 + 1 \times 2^4 + 0 \times 2^5 + 1 \times 2^6 + 1 \times 2^7 \\ &= 1 + 4 + 16 + 64 + 128 \\ &= (213)_{10} \end{aligned}$$

المثال (2-7): حول العدد $(11010101)_2$ إلى النظام الثماني octal، ثم إلى النظام الست عشري hex.

الحل:

$$(\underline{011} \underline{010} \underline{101})_2 = (325)_8$$

$$(\underline{1101} \underline{0101})_2 = (D5)_{16} = (D5)_{\text{hex}} = 0xD5$$

المثال (2-8): حول العدد $(69)_{\text{hex}}$ إلى النظام الثنائي ثم الثماني.

الحل:

$$(69)_{\text{hex}} = (0110 \ 1001)_2$$

$$(\underline{001} \underline{101} \underline{001})_2 = (151)_8$$

ملاحظة: يبدأ تجميع ثلاث خانات ثنائية عند التحويل من ثنائي إلى ثماني، وتجميع أربع خانات عند التحويل من ثنائي إلى الست عشري بدءاً من اليمين، مع إضافة أصفار على اليسار لإتمام المجموعة المؤلفة من 3 أو 4 خانات ثنائية.

المثال (2-9): حول العدد $(247)_{10}$ إلى النظام الست عشري hex.

الحل:

$$247 \div 16 = 15 \quad \text{rem } \underline{7}$$

$$15 \div 16 = 0 \quad \text{rem } \underline{15} = \underline{F} \quad \Rightarrow (247)_{10} = 0xF7$$

ويمكن التأكد من صحة الناتج بتحويله إلى النظام العشري كما يلي:

$$0xF7 = 7 \times 1 + F \times 16 = 7 + 15 \times 16 = 7 + 240 = (247)_{10}$$

المثال (2-10): أوجد المقابل العشري للأعداد الصحيحة الممثلة ثنائياً على

3 bit، بالطرق الثلاثة لتمثيل الأعداد السالبة: المطال والإشارة، المتمم الأحادي، المتمم الثنائي.

الحل:

المتمم الثنائي	المتمم الأحادي	المطال والإشارة	تمثيل العدد ثنائياً
0+	0+	0+	000
1+	1+	1+	001
2+	2+	2+	010
3+	3+	3+	011
4-	3-	0-	100
3-	2-	1-	101
2-	1-	2-	110
1-	0-	3-	111

المثال (11-2): مثل العدد $(-30)_{10}$ ثنائياً على 8 بت بطريقة:

أ- المطال والإشارة. ب- المتمم الأحادي. ج- المتمم الثنائي.

الحل:

أ- المطال والإشارة: $(-30)_{10} = 1001\ 1110$

ب- المتمم الأحادي: $(-30)_{10} = 1110\ 0001$

ج- المتمم الثنائي: $(-30)_{10} = 1110\ 0010$

المثال (12-2): حوّل العدد التالي الممثل عشرياً (-243) إلى التمثيل

الثنائي على 16 بت في الحالات التالية:

1- بطريقة تمثيل الأعداد الصحيحة بدون إشارة.

2- بطريقة المطال والإشارة.

3- بطريقة المتمم الأحادي.

4- بطريقة المتمم الثنائي.

الحل:

البت	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
بدون إشارة	غير ممكن															
مطال وإشارة	1	1	0	0	1	1	1	1	0	0	0	0	0	0	0	1
إتمام أحادي	0	0	1	1	0	0	0	0	1	1	1	1	1	1	1	1
إتمام ثنائي	1	0	1	1	0	0	0	0	1	1	1	1	1	1	1	1

المثال (2-13): ما هو مجال تمثيل الأعداد الصحيحة ثنائياً على 8 بت

بطريقة المطال والإشارة، الإتمام الثنائي؟

الحل:

أ- بطريقة المطال والإشارة:

المجال ثنائياً هو: 0111 1111... 11111111

ويقاله عشرياً: $127_{10} \dots 127_{10} -$. مع ملاحظة وجود تمثيلين للصفر

موجب وسالب: 00000000 ، 10000000، والبت الأعلى محجوز لإشارة العدد.

ب- بالإتمام الثنائي: المجال ثنائياً هو: 0111 1111... 10000000

ويقاله عشرياً: $127_{10} + \dots 128_{10} -$. مع ملاحظة وجود تمثيل وحيد

للصفر 00000000، وأن البت الأعلى يؤشر فقط على إشارة العدد وليس محجوزاً

لها. ومن أجل التمثيل على N بت يكون مجال الأعداد بطريقة الإتمام الثنائي هو:

$$-1 - 2^{(n-1)} \dots - 2^{(n-1)}.$$

المثال (2-14): ما هو التمثيل الثنائي لكل من العددين التاليين: $+127_{10}$ ، -128_{10} على 16 بت، بطريقة المتمم الثنائي؟

الحل:

هو نفس تمثيلهما الثنائي على 8 بت الوارد في المثال السابق، مع تمديدهما بالإشارة Sign Extension ليصبحا على 16 بت، فإن كان العدد سالباً فإن بت الإشارة 1 وبالتالي يمدد القسم الأعلى بنسخ الواحدات حتى يصبح على 16 بت، وإن كان العدد موجباً فيمدد بإضافة الأصفار حتى يصبح على 16 بت، وبالتالي يكون تمثيل العددين المطلوبين كما يلي:

$$+127_{10} = (00000000 \ 01111111)_2$$

$$-128_{10} = (11111111 \ 10000000)_2$$

المثال (2-15): ما هو التمثيل العشري للعدد التالي $(10011111)_2$ الممثل ثنائياً على 8 بت، بطريقة المتمم الثنائي ؟

الحل:

العدد المعطى سالب، وبالتالي لاستنتاج قيمته يلزم إيجاد المتمم الثنائي له:

$$1001 \ 1111 \Rightarrow 0110 \ 0000 \ (+1) \Rightarrow 0110 \ 0001$$

وبالتحويل إلى النظام العشري نجد:

$$2^6 + 2^5 + 1 = 97_{10}$$

وبالتالي فالعدد المعطى هو تمثيل للعدد السالب $(-97)_{10}$.

المثال (2-16): ما هو تمثيل العدد (-2.5) حسب المعيار IEEE754 بالدقة

المفردة؟

الحل:

$$0.5 = 1 \times 2^{-1} = 0.1 \quad , \quad 2 = 1 \times 2^{+1} = 10$$

$$\Rightarrow 2.5 = 10.1 = 1.01 \times 2^1$$

$$\text{exp} = p + \text{bias} = 1 + 127 = 128 = 10000000$$

$$s = 0$$

$$f = 010 \dots 0$$

فيكون تمثيل العدد (-2.5) حسب المعيار IEEE754 بالدقة المفردة هو:

$$1 \quad 10000000 \quad 010000000000000000000000$$

المثال (2-17): ما هو تمثيل العدد العشري (347.625) حسب المعيار

IEEE754 بالدقة المفردة؟

الحل:

$$347.625 = 101011011.101_2$$

ثم يلزم كتابة العدد الناتج بالشكل النظامي 1.f، أي بتحريك الفاصلة لإبقاء الرقم 1 فقط على يسار الفاصلة، وهو ما يسمى بالاستنظام Normalization:

$$101011011.101 \times 2^0 = 1.01011011101 \times 2^8 \Rightarrow$$

$$f = 01011011101 \quad , \quad s = 0$$

$$\text{exp} = P + 127 = 8 + 127 = 135 = 10000111$$

$$347.625 = 0 \quad 10000111 \quad 010110111010000000000000$$

المثال (2-18): ما هو تمثيل العدد $(-0.75)_{10}$ حسب المعيار IEEE754

بالدقة المفردة، وبالدقة المضاعفة؟

الحل:

$$-0.75 = -0.11 = -1.1 \times 2^{-1}$$

$$S=1$$

$$f=0.10\dots 0$$

$$\text{exp} = P+127 = -1+127 = 126 \quad (\text{بالدقة المفردة})$$

$$\text{exp} = P+1023 = -1+1023 = 1022 \quad (\text{بالدقة المضاعفة})$$

تمثيل العدد حسب المعيار IEEE754 بالدقة المفردة (32 بت):

$$1 \quad 0111 \ 1110 \quad 100 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000$$

$$1 \quad 0111 \ 1111 \ 110 \quad 10\dots 0 \quad : (\text{64 بت})$$

$$(-) \quad (\text{exp}=1022) \quad (f=52 \text{ bit})$$

المثال (2-19): ما هي القيمة العشرية المقابلة لهذا العدد الممثل حسب المعيار

IEEE754 بالدقة المفردة؟

$$1 \quad 01111100 \quad 110000000000000000000000$$

الحل:

نلاحظ أن $S=1$ فالعدد سالب.

قيمة الأس المنحاز $\text{exp}=124$ ، فالأس الحقيقي: $P=\text{exp}-127=-3$

الجزء الكسري $f=(0.11)_2$ ويقابل عشرياً: $f=0.5 + 0.25 = 0.75$

$$\Rightarrow -1.75 * 2^{-3} = -1.75 / 8 = -0.21875$$

تمارين غير محلولة:

التمرين (1-2): حوّل كلاً من العددين التاليين الممثلين ثنائياً إلى النظام العشري: $(100,00001)_2$, $(10011.111)_2$

التمرين (2-2): حوّل كلاً من العددين التاليين الممثلين عشرياً إلى النظام الثنائي: $(98,321)_{10}$, $(65.14)_{10}$

التمرين (3-2): حوّل العدد التالي الممثل بالنظام الست عشري $0x1A4$ إلى النظام الثنائي، الثماني، العشري.

التمرين (4-2): حوّل العدد التالي $(215)_{10}$ إلى النظام الست عشري.

التمرين (5-2): ما تمثيل كل من الأعداد العشرية التالية: -1 ، -6 ، $+6$ بكل من طريقة الإتمام الثنائي، وطريقة المطال والإشارة، على 4 بت ثم على 8 بت؟

التمرين (6-2): ليكن لدينا العدد التالي الممثل ثنائياً على 16 بت:

1000000000011111

ما هي القيمة العشرية للعدد بفرض أن طريقة التمثيل:

أ- بالمطال والإشارة. ب- بالإتمام الثنائي. ج- بدون إشارة.

التمرين (7-2): ما هو تمثيل كل من العددين $(2, 0.5)$ حسب المعيار IEEE754 بالدقة المفردة؟

التمرين (8-2): ما هو تمثيل العدد (-175) حسب المعيار IEEE754 بالدقة المفردة؟ ثم بالدقة المضاعفة؟

التمرين (9-2): ما القيمة العشرية المقابلة للعدد التالي الممثل حسب المعيار IEEE754 دقة مفردة: 0 10000001 010110000000000000000000 ؟

الفصل الثالث

الحساب في الحاسوب

مقدمة:

يتم الاعتماد في العمليات الحسابية التي تجري في الحاسوب بشكل أساسي على طريقة تمثيل الأعداد ثنائياً، وعلى الخوارزميات المستخدمة في العمليات الحسابية (جمع، طرح، ضرب، قسمة)، سواء كانت الأعداد صحيحة، أو أعداد ذات فاصلة عائمة. ويمكن إجراء العمليات الحسابية بأنواعها كما هو الحال في النظام العشري مع مراعاة أن أساس النظام المستعمل هنا هو 2.

عند إجراء العمليات الحسابية قد ينتج حالة فيض overflow أو غيض underflow، ويحدث الفيض إذا كانت القيمة كبيرة جداً بحيث لا يمكن تمثيلها بالتمثيل المعطى، وهذه القيمة قد تكون موجبة أو سالبة. بينما يحدث الغيض إذا كانت القيمة صغيرة جداً إلى حد لا يسمح بتمثيلها، سواء كانت موجبة أو سالبة.

يمكن إجراء عملية الضرب في النظام الثنائي على الأعداد الممثلة بالمطال والإشارة أو بالإتمام الأحادي أو الثنائي. وتعتبر طريقة الضرب باستخدام الأعداد الممثلة بالمطال والإشارة الطريقة المثلى في حالتي الضرب والقسمة لسهولة التعامل مع الإشارة، حيث أن ضرب وقسمة أي عددين مختلفين في الإشارة يعطي نتيجة سالبة، بينما ضرب وقسمة أي عددين متشابهين في الإشارة سواء كانت سالبة أو موجبة يعطي نتيجة موجبة. وتعتبر عملية الضرب سلسلة من عمليات الجمع المتتالي والإزاحة، بينما تعتبر عملية القسمة سلسلة من عمليات الطرح المتتالي والإزاحة.

يتناول هذا الفصل أمثلة محلولة وتمارين عن العمليات الحسابية على الأعداد الموجبة (بدون إشارة unsigned)، وعلى الأعداد الصحيحة الممثلة بطريقة الإتمام الثنائي، بالإضافة إلى العمليات الحسابية على الأعداد ذات الفاصلة العائمة.

أمثلة محلولة:

المثال (3-1): أوجد ناتج العمليات على الأعداد الموجبة unsigned الممثلة على 6 بت.

الحل:

$(10) + (14)$	$(33) + (29)$	$(11) - (7)$
001010	100001	001011
<u>001110</u>	<u>011101</u> +	<u>000111</u> -
011000	111110	000100
(24)	(62)	(4)

المثال (3-2): أوجد ناتج جمع الأعداد التالية الممثلة بطريقة الالتزام الثنائي على 8 بت:

$(5+12)$ ، $(-5+12)$ ، $(-12+-5)$ ، $(12+-12)$.

الحل:

$5+12$	$(-5)+12$	$-12+(-5)$	$12+(-12)$
00000101	11111011	11110100	00001100
<u>00001100</u>	<u>00001100</u>	<u>11111011</u>	<u>11110100</u>
00010001	00000111	11101111	00000000
17	7	-17	0

المثال (3-3): أوجد تمثيل كل من العددين 6 ، 7 ثنائياً على 32 بت ثم أوجد ناتج طرحهما.

الحل:

إما أن يتم الطرح بشكل مباشر كما في طرح الأعداد بالنظام العشري:

$$\begin{array}{r} 7_{10} = 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0111 \\ -6_{10} = 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0110 \\ \hline 1_{10} = 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001 \end{array}$$

أو بجمع العدد 7 مع المتمم الثنائي للعدد 6:

$$\begin{array}{r} 7_{10} = 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0111 \\ -6_{10} = 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1010 \\ \hline 1_{10} = 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001 \end{array}$$

المثال (3-4): أوجد التمثيل الثنائي لكل من العددين 13 ، 11 بدون إشارة على 4 بت، ثم أوجد ناتج ضربيهما ثنائياً.

الحل:

$$\begin{array}{r} 1101 \quad (13) \\ \times 1011 \quad (11) \\ \hline 1101 \\ 1101 \\ 0000 \\ 1101 \quad + \\ \hline 10001111 \quad (143) \end{array}$$

نلاحظ أن ناتج ضرب عددين ثنائيين كل منهما مؤلف من 4 بت يلزم تمثيله على 8 بت.

المثال (3-5): أوجد ناتج قسمة العدد $(1101)_2$ على $(10)_2$.
الحل:

$$\begin{array}{r}
 110.1 \\
 10 \overline{) 1101} \\
 \underline{10-} \\
 10 \\
 \underline{10-} \\
 010 \\
 \underline{10-} \\
 00
 \end{array}
 \qquad
 \begin{array}{r}
 110 \\
 10 \overline{) 1101} \\
 \underline{10-} \\
 10 \\
 \underline{10-} \\
 01
 \end{array}$$

أي يمكن التعبير عن حاصل قسمة 1101 على 10 ثنائياً بأن الناتج هو 110_2 والباقي 1، أو بالشكل 110.1_2 ، ويقابله عشرياً أن حاصل قسمة 13 على 2 هو 6 والباقي 1 أو (6.5) .

المثال (3-6): أوجد ناتج جمع كل من العددين 9 و 8 بعد تحويلهما إلى أعداد ثنائية بدون إشارة على 4 بت، هل الناتج صحيح؟
الحل:

$$\begin{array}{r}
 (8) \quad 1000 \\
 (9) \quad 1001+ \\
 \hline
 (1) \quad 0001
 \end{array}$$

نلاحظ أن حاصل الجمع على 4 بت هو 1 وهو جواب غير صحيح، وإنما

يجب أن يكون 17، ولكن لا يمكن تمثيل العدد 17 على 4 بت، وتسمى هذه الحالة بالفيض overflow لأن الناتج خارج مجال تمثيل الأعداد وهو في حالتنا [0..15]، ويمكن الاستدلال على وجود الفيض في العمليات على الأعداد بدون إشارة unsigned من وجود الحمل carry من الخانة العليا Cout=1 كما في هذا المثال، أما إذا كان Cout=0 فالناتج صحيح.

المثال (3-7): أوجد ناتج جمع الأعداد التالية الممثلة ثنائياً بدون إشارة، على 8 بت، هل يوجد فيض؟

الحل:

0000 0001 (1 ₁₀)	0010 1100 (44 ₁₀)
1111 1111 (255 ₁₀)	0101 0101 (85 ₁₀)
0000 0000 (0 ₁₀)	1000 0001 (129 ₁₀)
Cout=1 (overflow)	Cout=0 (no overflow)

المثال (3-8): أوجد ناتج جمع كل من العددين 3 و 6 بعد تحويلهما إلى أعداد ثنائية، على 4 بت، وبالإتمام الثنائي هل الناتج صحيح؟

الحل:

$$\begin{array}{r}
 (3) \quad 0011 \\
 (6) \quad 0110 + \\
 \hline
 (7-) \quad 1001
 \end{array}$$

نلاحظ أن ناتج جمع عددين موجبين أعطى عدداً سالباً هو -7 فالناتج غير صحيح، ويتم الاستدلال على حصول الفيض في حالة الإتمام الثنائي إذا كانت قيمة الحمل الداخلة إلى الخانة العليا Cin لا تساوي قيمة الحمل الناتجة عن الخانة العليا

Cout، وفي مثالنا هذا نجد عند جمع العددين بأن Cin للبت الرابع هو 1 و Cout له هو 0، وهذا دليل على وجود فيض وبأن الناتج غير صحيح، كما يمكن ملاحظة ذلك من أن مجال تمثيل الأعداد على 4 بت في طريقة الإتمام الثنائي هو: [-8..7] وأن ناتج جمع 3 مع 6 يقع خارج هذا المجال.

المثال (3-9): أوجد ناتج جمع الأعداد التالية الممثلة ثنائياً، على 8 بت، وبالإتمام الثنائي، هل يوجد فيض؟

$$101100+1010101, \quad 111111+11010101, \quad 10111101+1110 \ 0101$$

الحل:

1011 1101 (-67_{10})	0011 1111 (63_{10})	0010 1100 (44_{10})
<u>1110 0101 (-27_{10})</u>	<u>+1101 0101 (-43_{10})</u>	<u>+0101 0101 (85_{10})</u>
1010 0010 (-94_{10})	0001 0100 (20_{10})	1000 0001 (-127_{10})
(no overflow)	(no overflow)	(overflow)
Cin=1, Cout=1	Cin=1, Cout=1	Cin=1, Cout=0

المثال (3-10): أوجد ناتج طرح العدد 00011011_2 من العدد 00110001_2 ، علماً أن الأعداد ممثلة ثنائياً بالإتمام الثنائي على 8 بت، هل يوجد فيض؟

الحل: يتم تحويل عملية الطرح إلى عملية جمع مع المتمم الثنائي:

$$\begin{array}{r}
 0011 \ 0001 \\
 - 0001 \ 1011 \Rightarrow + 1110 \ 0101 \\
 \hline
 0001 \ 0110 \text{ (Cin=Cout=1} \Rightarrow \text{no overflow)}
 \end{array}$$

المثال (3-11): أوجد ناتج جمع العددين التاليين، وافترض أن الدقة precision 4 بت (أو الجزء الكسري f=3bit):

$$(1.111)_2 \times 2^{-1}, (1.011)_2 \times 2^{-3}$$

الحل:

1- يلزم توحيد الأسس حتى نتمكن من جمع الأعداد، يتم إزاحة العدد ذي الأس الأصغر إلى اليمين إلى أن يساوي قيمة الأس الأكبر:

$$(1.011)_2 \times 2^{-3} = (0.1011)_2 \times 2^{-2} = (0.01011)_2 \times 2^{-1}$$

بما أن الفرق بين الأسين هو 2 تتم إزاحة العدد ذي الأس الأصغر إلى اليمين بمقدار 2.

2- جمع العددين بعد أن تم توحيد الأسس:

$$\begin{array}{r} 1.111 \\ +0.01011 \\ \hline 10.00111 \end{array}$$

3- نلاحظ أن الناتج غير نظامي، ولذا يلزمه استنظام ليصبح بالشكل 1.f:

$$(10.00111)_2 \times 2^{-1} = (1.000111)_2 \times 2^0 = (1.000111)_2$$

بما أنه تم إزاحة الفاصلة بمقدار خانة واحدة لليمين فيتم زيادة الأس بمقدار 1.

4- تدوير Rounding العدد للأقرب حسب الدقة المطلوبة، وفي مثالنا f=3:

$$(1.000111)_2 \approx (1.001)_2$$

5- بعد التدوير إذا نتج العدد بشكل غير نظامي فيلزم إعادة الاستنظام له.

6- اختبار حصول الفيض overflow أو الغيظ underflow (هل الأس

الناتج كبير جداً (فيض)، أو صغير جداً (غيظ)، وذلك من خلال العلاقة:

$$-126 \leq (P=0) \leq 127$$

$$1 \leq (\text{Exp}=127) \leq 254$$

وبالتالي لا يوجد فيض أو غيض. وناتج الجمع النهائي هو: $(1.001)_2$.

ملاحظة: في حال كانت إشارة العددين المراد جمعها نفسها فعندها يُجمع العددان وتكون إشارة الناتج هي نفس إشارة العددين، أما في حال كانت الإشارتان مختلفتين فيتم طرح العدد الأصغر من العدد الأكبر وتكون إشارة الناتج هي نفس إشارة العدد الأكبر بينهما.

المثال (3-12): أوجد ناتج طرح العدد 0.4375 من العدد 0.5 بعد تحويلهما إلى أعداد ثنائية، وافترض أن الدقة precision 4 بت (f=3bit):
الحل:

1- تحويل العددين من النظام العشري إلى الثنائي:

$$0.5 = 0.1 \times 2^0 = 1.0 \times 2^{-1} \text{ (normalised)}$$

$$-0.4375 = -0.0111 \times 2^0 = -1.110 \times 2^{-2} \text{ (normalised)}$$

2- توحيد الأسس: بإزاحة العدد ذي الأس الأصغر إلى اليمين إلى أن يساوي قيمة الأس الأكبر (الفرق بين الأسين $1 - (-2) = 1 - (-2) = 1$ فتكون الإزاحة=1):

$$-1.110 \times 2^{-2} = -0.1110 \times 2^{-1}$$

3- طرح العددين بعد أن تم توحيد الأسس:

$$\begin{array}{r} 1.0000 \times 2^{-1} \\ - 0.1110 \times 2^{-1} \\ \hline 0.0010 \times 2^{-1} \end{array}$$

4- نلاحظ أن الناتج غير نظامي، ولذا يلزمه استتظام ليصبح بالشكل 1.f:

$$0.0010 \times 2^{-1} = 1.000 \times 2^{-4}$$

5- تدوير العدد للأقرب حسب الدقة المطلوبة، وفي مثالنا $f=3$:
 نلاحظ أن الناتج $2^{-4} \times 1.000$ يناسب الدقة 4 بت وبالتالي لا حاجة للتدوير.
 6- اختبار حصول الفيض أو الغيض، وذلك من خلال العلاقة:
 $127 \geq (p=-4) \geq -126$ ، وبالتالي نجد أنه لا يوجد فيض أو غيض.
 ملاحظة: يمكن التحقق من صحة الناتج بإعادة تحويله إلى النظام العشري
 فنجد: $1.000 \times 2^{-4} = 0.0625$ وهو يساوي ناتج عملية الطرح $0.5 - 0.4375$ ،
 فالناتج صحيح.

المثال (3-13): أوجد ناتج ضرب العددين 1.000×2^{-1} ، -1.110×2^{-2} ،
 ثنائياً، علماً أن الدقة 4 بت.

الحل:

1- نجمع الأسين، ونوجد الأس المنحاز biased exponents:

$$(-1) + (-2) + 127 = -3 + 127 = 124$$

2- نضرب العددين ثنائياً:

$$\begin{array}{r} 1.000 \\ \times 1.110 \\ \hline 0000 \\ 1000 \\ 1000 \\ + 1000 \\ \hline \end{array}$$

$$1110000 \Rightarrow 1.110000$$

إذاً ناتج الضرب هو 1.110×2^{-3} (من أجل الدقة 4 بت)

- 3- استنظام الناتج: نجد أنه مكتوب بالشكل النظامي normalised .
- 4- تدوير الناتج حسب الدقة المطلوبة : لا حاجة للتدوير في هذا المثال.
- 5- اختبار الغيض/ الفيض: $127 \geq -3 \geq -126$ فالناتج صحيح.
- 6- تحديد إشارة الناتج: بما أن العددين لهما إشارتين مختلفتين فإن الناتج عدد سالب (إجراء XOR بين بتي الإشارة للعددين)، وبالتالي نكتب ناتج الضرب بالشكل:
- $$-1.110 \times 2^{-3}$$

المثال (3-14): أوجد ناتج ضرب العددين التاليين، ثم مثل الناتج النهائي حسب المعيار IEEE754 دقة مفردة.

$$\begin{aligned} & -1.110\ 1000\ 0100\ 0000\ 1010\ 0001_2 \times 2^{-4} \\ & \times \quad 1.100\ 0000\ 0001\ 0000\ 0000\ 0000_2 \times 2^{-2} \end{aligned}$$

الحل:

1- نجمع الأسين، ونوجد الأس المنحاز: $P = (-4) + (-2) = -6$

2- نضرب العددين ثنائياً:

$$\begin{array}{r} 1.11010000100000010100001 \\ \times \quad 1.10000000001000000000000 \\ \hline 111010000100000010100001 \\ 111010000100000010100001 \\ + 111010000100000010100001 \\ \hline 10101110001111101111100110010100001000000000000 \end{array}$$

وموضع الفاصلة بعد 46 بت بدءاً من اليمين، وبالتالي فإن ناتج ضرب العددين

هو:

$$10.1011100011111011111001100101000010000000000000 \times 2^{-6}$$

3- استنظام الناتج، بالإزاحة لليمين بمقدار 1 وزيادة قيمة الأس بمقدار 1:

$$-1.0101110001111101111100110010100001 \times 2^{-5}$$

4- تدوير الناتج بحيث يكون f=23 bit

$$1.01011100011111011111101 \times 2^{-5}$$

5- اختبار الغيض/ الفيض: $127 \leq -5 \leq -126$ فالناتج صحيح.

6- تحديد إشارة الناتج: بما أن العددين لهما إشارتين مختلفتين فإن الناتج عدد سالب، وبالتالي نكتب ناتج الضرب بالشكل:

$$-1.01011100011111011111101 \times 2^{-5}$$

7- نحسب الأس المنحاز: $\text{exp} = -5 + 127 = 122$

8- تمثيل الناتج حسب المعيار IEEE 754 بدقة مفردة:

1	0	1	1	1	1	0	1	0	0	1	0	1	1	1	0	0	0	1	1	1	1	1	0	1	1	1	1	1	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

المثال (3-15): أوجد ناتج قسمة العدد 127.03125 على العدد 16.9375

بعد تمثيلهما ثنائياً، ثم مثّل كلا من العددين والناتج النهائي حسب المعيار IEEE754 بدقة مفردة.

الحل:

1- نمثّل كلا من العددين ثنائياً:

$$127.03125 = 1111111.00001 = 1.11111100001 \times 2^6 \quad (\text{exp}=133)$$

$$16.9375 = 10000.1111 = 1.00001111 \times 2^4 \quad (\text{exp}=131)$$

2- نقسم العددين ثنائياً: (تم إلغاء الفاصلة من العددين بإزاحتها بمقدار 11

خانة):

$$\begin{array}{r}
 1.111 \\
 \hline
 100001111000 \quad 111111100001 \\
 \hline
 100001111000 - \\
 \hline
 0111011010010 \\
 \hline
 100001111000 - \\
 \hline
 00110010110100 \\
 \hline
 100001111000 - \\
 \hline
 0100001111000 \\
 \hline
 100001111000 - \\
 \hline
 00000000000000
 \end{array}$$

3- نوجد إشارة حاصل القسمة، بما أن العددين موجبين فالناتج موجب، أي أن بت الإشارة = 0.

4- نحسب الأس للعدد الناتج، وهو ناتج طرح الأسين، ونوجد الأس المنحاز:

$$P = 6 - 4 = 2 \Rightarrow$$

$$\text{exp} = 2 + 127 = 129 = 10000001$$

5- الاستنتظام: نلاحظ أن حاصل القسمة بالشكل النظامي: 1.111×2^2

6- اختبار الغيض/الفيض: $127 \leq 2 \leq -126$ فالناتج صحيح.

7- تمثيل العددين وحاصل القسمة حسب المعيار IEEE754 بدقة مفردة:

127.03125	0	10000101	111 1110 0001 0000 0000 0000
-----------	---	----------	------------------------------

16.9375	0	10000011	000 0111 1000 0000 0000 0000
---------	---	----------	------------------------------

حاصل القسمة	0	10000001	111 0000 0000 0000 0000 0000
-------------	---	----------	------------------------------

ملاحظة: يمكن التحقق من صحة الحل بإجراء قسمة العدد 127.03125 على العدد 16.9375 فنجد أن الناتج هو 7.5 وتمثيله الثنائي: 1.111×2^2 فالحل صحيح.

تمارين غير محلولة:

التمرين (1-3): مثل كلاً من العددين 128_{10} ، 200_{10} ثنائياً على 8 بت بدون إشارة unsigned، ثم أوجد ناتج جمعهما، وطرحهما، هل الناتج صحيح؟

التمرين (2-3): أوجد ناتج جمع الأعداد التالية بعد تمثيلها ثنائياً بطريقة الإتمام الثنائي على 8 بت، وهل الناتج صحيح في كل منها؟

$$(80 + 120)، (-80 + -120)، (-80 + 120)، (80 + -12)$$

التمرين (3-3): أوجد التمثيل الثنائي لكل من العددين 13، 42 بدون إشارة على 6 بت، ثم أوجد ناتج ضربهما ثنائياً، وحاصل قسمة 42 على 13 ثنائياً.

التمرين (4-3): أوجد ناتج جمع ثم طرح كل من العددين 25 و 18 بعد تحويلهما إلى أعداد ثنائية، على 6 بت، وبالإتمام الثنائي، هل الناتج صحيح؟

التمرين (5-3): أوجد ناتج طرح العدد 1111_2 من العدد 0111_2 ، علماً أن الأعداد ممثلة ثنائياً بالإتمام الثنائي على 4 بت، هل يوجد فيض؟

التمرين (6-3): أوجد التمثيل الثنائي لكل من العددين 14، 5.75 حسب المعيار IEEE754 دقة مفردة، ثم أوجد ناتج جمعهما بفرض أن الدقة 4 بت.

التمرين (7-3): أوجد ناتج جمع العددين التاليين ومثل الناتج حسب المعيار IEEE754 دقة مفردة:

$$1.11100100000000000000010_2 \times 2^4 \\ + 1.100000000000000110000101_2 \times 2^2$$

التمرين (8-3): أوجد التمثيل الثنائي للعددين 0.5، 0.4375 حسب المعيار IEEE754 دقة مفردة، ثم أوجد ناتج جمعهما، بفرض أن الدقة 4 بت.

التمرين (3-9): أوجد ناتج طرح العدد التالي $2^2 \times (1.000)_2$ من العدد $2^{-3} \times (1.000)_2$ ، بفرض أن الدقة 4 بت (f=3bit).

التمرين (3-10): اكتب التمثيل الثنائي للعددين التاليين: 17، 23.125، ثم أوجد ناتج جمعهما، ثم ناتج طرحهما، مع تمثيل الناتج حسب المعيار IEEE754 دقة مفردة، بفرض أن الدقة 5 بت.

التمرين (3-11): أوجد ناتج ضرب العددين التاليين، بفرض أن الدقة 4 بت:

$$1.010_2 \times 2^{-1} \times -1.110_2 \times 2^{-2}$$

التمرين (3-12): اكتب التمثيل الثنائي للعددين التاليين: 3.75، -5.25، ثم أوجد ناتج ضربيهما، بفرض أن الدقة 5 بت.

التمرين (3-13): أوجد ناتج قسمة العدد 1011.11_2 على العدد 11_2 ، ومثل ناتج القسمة حسب المعيار IEEE754 دقة مفردة، بفرض أن الدقة 6 بت.

التمرين (3-14): اكتب التمثيل الثنائي للعددين التاليين: 21.5، 4، ثم أوجد ناتج قسمة العدد الأول على الثاني، ومثل ناتج القسمة حسب المعيار IEEE754 دقة مفردة، بفرض أن الدقة 5 بت.

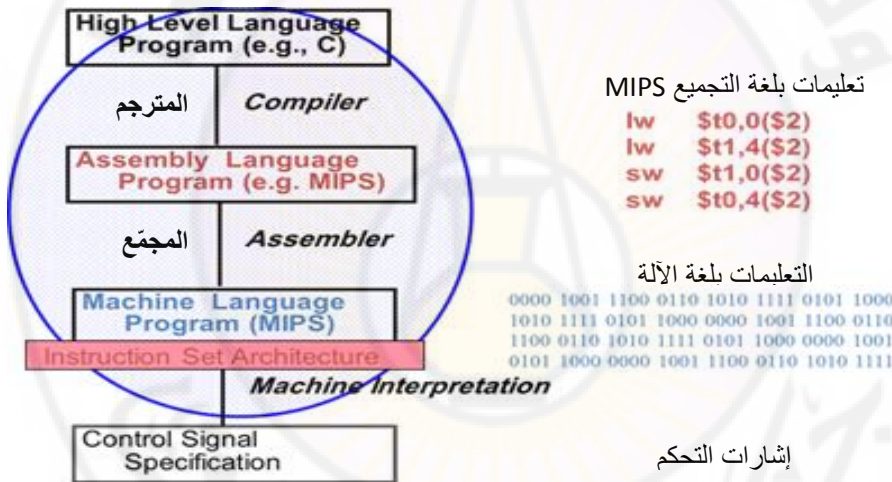


الفصل الرابع

البرمجة بلغة التجميع MIPS

مقدمة:

لدينا بشكل عام ثلاثة مستويات للبرمجة وهي: اللغات العالية المستوى HLL، البرمجة بلغة التجميع Assembly، البرمجة بلغة الآلة Machine Language، وفي هذا الفصل سنتناول البرمجة بلغة التجميع الخاصة بمعالجات MIPS.



تستخدم لغة التجميع السجلات كمعاملات للتعليمات بدلاً من المتحولات، ويحوي معالج MIPS الذي سندرسه في منهاجنا 32 سجلاً للأعداد الصحيحة بعرض 32 بت، بالإضافة إلى سجل عداد البرنامج PC والسجلين HI, LO المستخدمين في عمليات الضرب والقسمة، بالإضافة إلى سجلات الأعداد ذات الفاصلة العائمة.

تتألف كل تعليمة من 32 بت، ولدينا ثلاثة أنماط للتعليمات في لغة التجميع

MIPS وهي كالتالي:

-النمط R_Type: وتتألف التعليمات فيه من الحقول التالية:

R_Type	OP (6 bit)	Rs (5)	Rt (5)	Rd (5)	S.A (5)	Funct
--------	------------	--------	--------	--------	---------	-------

ومن تعليمات هذا النمط: add, sub, and, or, slt, sll, sra.

-النمط I_Type: وتتألف التعليمات فيه من الحقول التالية:

I_Type	OP (6 bit)	Rs	Rt (5)	Immediate (16 bit)
--------	------------	----	--------	--------------------

ومن تعليماته: addi, ori, andi, slti, lw, sw, beq, bne.

-النمط J_Type: وفي هذا النمط تعليمتان هما: J, Jal. وتتألف التعليمات فيه

من حقلين فقط كما يلي:

J_Type	OP (6 bit)	jump target (26)
--------	------------	------------------

بالإضافة إلى التعليمات الحقيقية من الأنماط السابقة والتي يمكن تنفيذها عتادياً في معالج MIPS، يوجد تعليمات زائفة Pseudo Instructions لا يتم تنفيذها عتادياً، وإنما يتولى المجمع assembler تحويلها إلى تعليمات حقيقية مكافئة، ومثال عنها التعليمات move, la, li, bgt, blt, bge, ble, abs, mul وغيرها، وسيتم التعرف على التعليمات الحقيقية والزائفة واستخدامها من خلال الأمثلة المحولة في هذا الفصل.

أما بالنسبة للذاكرة فلها مسرى عناوين مؤلف من 32 بت، ومجال العناوين الممكنة هو من 0x00000000 وحتى 0xFFFFFFFF، يتم تحميل كلمة من الذاكرة باستخدام التعليمات lw rt, offset(rs) حيث يسمى السجل rs بالسجل القاعدي base register، ومحتواه عبارة عن مؤشر إلى الذاكرة، يضاف إليه قيمة انزياح عددي offset للحصول على عنوان الذاكرة، فيتم تحميل كلمة (32 بت) من هذا العنوان المحسوب إلى السجل rt.

بينما تعليمة تخزين كلمة في الذاكرة هي $sw\ rt, offset(rs)$ ، ويتم حساب عنوان الذاكرة كما في تعليمة lw تماماً، ويتم تخزين محتوى السجل rt في الذاكرة عند هذا العنوان. أما لتحميل أو تخزين بايت واحد فقط فتُستخدم التعليمتان lb, sb كما في حالة التعامل مع المحارف.

يمكن أن تكون عنوان الذاكرة في المعالجات بالكلمات، ولكن غالباً تكون العنوان بالبايت، أي لكل بايت عنوان خاص به مؤلف في حالتنا من 32 بت، وبما أن الكلمة مؤلفة من 4 بايت فإن عناوين كلمات الذاكرة من مضاعفات العدد 4.

عند ترجمة السطر البرمجي التالي في لغة C: $g=A[2]$; فإن مقدار الانزياح بالبايت في تعليمة تحميل الكلمة ذات الدليل 2 من المصفوفة A هو $4 \times 2 = 8$ ، فنستخدم تعليمة تحميل كلمة من الذاكرة كما يلي:

$lw\ \$t0, 8(\$s3)\ \#\ \$t0\ gets\ A[2]$

فإذا كان العنوان القاعدي $\$s3 = 0x10007000$ فإن عنوان العنصر $A[2]$ هو $0x10007008$ ، كما هو موضح في الشكل التالي:

Address	Data
0x10007010	array[4]
0x1000700C	array[3]
0x10007008	array[2]
0x10007004	array[1]
0x10007000	array[0]

Main Memory

ويمكن استخدام المكس $stack$ ، وهو عبارة عن مقطع في الذاكرة تُخزن فيه المتحولات المحلية $local\ variables$ ، والسجلات المحفوظة $saved\ registers$ ، ويستعمل السجل $\$sp$ كمؤشر للمكس $Stack\ pointer$.

لتخزين محتوى سجل العودة \$ra مثلاً في المكس، نستخدم التعليمتين التاليتين:

```
addi $sp,$sp, -4
```

```
sw $ra, 0($sp) # save $ra
```

ولاسترجاع هذه الكلمة من المكس نستخدم التعليمتين:

```
lw $ra, 0($sp) # restore $ra
```

```
addi $sp,$sp, +4
```



أمثلة محلولة:

المثال (1-4): اكتب برنامجاً بلغة التجميع MIPS لحساب: $y = 2a + b - 5$.

الحل: بفرض أن قيمة المتحول a مخزنة في السجل $t0$ ، وقيمة المتحول b في السجل $t1$ ، وأن المتحول y يقابل السجل $v0$:

```
add $t2,$t0,$t0    # t2=2a
add $t2,$t2,$t1     # t2= 2a + b
addi $v0, $t2, -5   # y= 2a+b-5
```

المثال (2-4): اكتب التعليمات الحقيقية المقابلة للتعليمات الزائفة التالية:

التعليمات الزائفة pseudo	التعليمات الحقيقية / التعليمة
move \$t0,\$t1	add \$8,\$0,\$9
li \$t0, 5000	addi \$8,\$0,5000
mul \$t0,\$t1,\$t2	mult \$t1,\$t2 mflo \$t0
div \$t0,\$t1,\$t2	div \$t1,\$t2 mflo \$t0
bge \$t0, \$s0, LABEL	slt \$at, \$t0, \$s0 beq \$at, \$zero, LABEL
blt \$t0, \$s0, LABEL	slt \$at, \$t0, \$s0 bne \$at, \$zero, LABEL
nop	sll \$0, \$0, 0

المثال (3-4): ما هي التعليمة الزائفة التي يمكن من خلالها شحن القيمة 0xAB12 في السجل \$t0؟ وما هي التعليمة الزائفة التي يمكن من خلالها شحن القيمة 0xABCD1234 في السجل \$t1؟ وما هي التعليمات الحقيقية المكافئة لكل منهما؟
الحل:

1) `li $t0, 0xAB12`

وبما أن العدد 0xAB12 يمكن تمثيله على 16 بت فتكون التعليمة الحقيقية المكافئة للتعليمة السابقة هي:

`ori $t0,$zero, 0xAB12`

2) `li $t1, 0xABCD1234`

بما أن العدد 0xABCD1234 مؤلف من 32 بت فلا يمكن تمثيله ضمن الحقل imm المؤلف من 16 بت في تعليمة ori، لذا يلزم استخدام التعليمة lui وهي من النمط l_type تأخذ 16 بت العليا من العدد وتضعها في النصف العلوي من السجل المعطى، وفي النصف الأدنى منه أصفاراً. وبالتالي فإن التعليمات الحقيقية المكافئة للتعليمة السابقة هي:

`lui $at, 0xABCD # $at = ABCD0000`

`ori $t1, $at, 0x1234 # ABCD0000 or 00001234`

`# 0xABCD1234 → $t1`

المثال (4-4): ما هي التعليمة الزائفة التي يمكن من خلالها تحميل العنوان label وقيمه 0x00400008 في السجل \$t0؟ وما هي التعليمات الحقيقية المكافئة لها؟

الحل:

التعليمة الزائفة لتحميل عنوان ذاكرة مكون من 32 بت في سجل هي:

la \$t0, label # address of label is 0x00400008 → \$t0

ويكافئها التعليمتان الحقيقيتان:

lui \$1, 0x40 # load upper(label) in \$at (0x0040)

ori \$8, \$1, 0x0008 # ori \$t0, \$at, lower(label);

(\$t0=0x0040 0008)

ملاحظة: الفرق بين التعليمتين السابقتين la, li أن التعليمة la تشحن السجل

بعنوان مؤلف من 32 بت كحد أعظمي، بينما التعليمة li تشحن السجل بمعطيات مؤلفة من 32 بت كحد أعظمي.

المثال (4-5): بفرض أن محتوى السجلين \$t0, \$t1 كما يلي:

\$t0= 0000 0000 0000 0000 1111 1111 1111 1111

\$t1= 1111 1111 1111 1111 0000 0000 0000 0000

ما هو محتوى كل من السجلات \$t2, \$t3, \$t4, \$t5 بعد تنفيذ التعليمات

التالية:

slt \$t2, \$t0,\$t1

sltu \$t3, \$t0,\$t1

slti \$t4, \$t1,0x7111

sltiu \$t5, \$t1,0x7111

الحل:

في التعليمة الأولى تتم المقارنة بين محتوى السجلين \$t0, \$t1 مع أخذ الإشارة

بعين الاعتبار، حيث أن محتوى \$t0 موجب بينما محتوى \$t1 سالب، لذا فإن

$t_0 > t_1$ وبالتالي فإن $t_2 = 0$.

أما في التعليمة الثانية فنتم المقارنة بين محتوى السجلين t_0 , t_1 باعتبارهما بدون إشارة unsigned، وأن كليهما موجب، لذا فإن $t_0 < t_1$ وبالتالي فإن $t_3 = 1$.
في التعليمة الثالثة $slti$ تتم المقارنة بين محتوى t_1 كعدد سالب مع العدد $0x7111$ بعد تمديده بالإشارة sign extending، وبالتالي فإن $t_4 = 1$.

أما في التعليمة الرابعة $sltiu$ فنتم المقارنة بين محتوى t_1 كعدد موجب مع العدد $0x7111$ ، فينتج أن $t_5 = 0$.

ملاحظة: في التعليمة $slti$ فإن كلاً من rs , imm عبارة عن أعداد صحيحة ممثلة بالإتمام الثنائي، أما في تعليمة $sltiu$ فهي عبارة عن أعداد صحيحة موجبة (بدون إشارة).

أما في التعليمات $addu$, $subu$, $addiu$ فإن الحرف u في نهاية التعليمة يشير إلى تجاهل الطفح ignore overflow.

المثال (4-6): بفرض أن محتوى السجل t_0 هو $0x12345678$ أوجد ناتج تنفيذ تعليمات الإزاحة التالية في سجلات الوجهة المحددة:

$sll \quad t_1, t_0, 8 \quad \# \quad \text{shift left logical } t_0 \text{ by 8 bits} \rightarrow t_1$

$srl \quad t_2, t_0, 8 \quad \# \quad \text{shift right logical by 8 bits} \rightarrow t_2$

الحل:

نوجد التمثيل الثنائي لمحتوى السجل t_0 :

$t_0 = \underline{0001 \ 0010 \ 0011 \ 0100 \ 0101 \ 0110 \ 0111 \ 1000}$

$t_1 = t_0 << 8 = 0011 \ 0100 \ 0101 \ 0110 \ 0111 \ 1000 \ 0000 \ 0000$

$t_1 = 0x34567800$

$\$t2 = \$t0 \gg 8 = 0000\ 0000\ 0001\ 0010\ 0011\ 0100\ 0101\ 0110$

$\$t2 = 0x00123456$

المثال (4-7): بفرض أن محتوى السجل $\$t0$ هو $0xFFFF\ 0000$ ، ومحتوى السجل $\$t1$ هو $0x0000\ FFFF$ أوجد ناتج تنفيذ تعليمة الإزاحة التالية في سجلات الوجهة المحددة:

$sra\ \$t3,\ \$t0, 3\ \# \text{ shift right arithmetic by 3 bits} \rightarrow \$t3$

$sra\ \$t4,\ \$t1, 3\ \# \text{ shift right arithmetic by 3 bits} \rightarrow \$t4$

الحل:

نوجد التمثيل الثنائي لمحتوى السجلين $\$t0, \$t1$:

$\$t0 = 1111\ 1111\ 1111\ 1111\ 0000\ 0000\ 0000\ 0000$

$\$t1 = 0000\ 0000\ 0000\ 0000\ 1111\ 1111\ 1111\ 1111$

وبالتالي فإن:

$\$t3 = 11111111\ 1111\ 1111\ 1110\ 0000\ 0000\ 0000\ 0000$

$\$t3 = 0xFFFFe000$

$\$t4 = 0000\ 0000\ 0000\ 0000\ 0001\ 1111\ 1111\ 1111$

$\$t4 = 0x00001FFF$

في هذه التعليمة تتم الإزاحة إلى اليمين مع ملء الفراغات بنفس بت الإشارة (0) أو (1).

المثال (4-8): اكتب بلغة التجميع MIPS التعليمات التي تكافئ السطر البرمجي التالي بلغة C:

$A[0] = C[0] \ll 4;$

الحل:

بفرض أن العنوان القاعدي للمصفوفة C هو محتوى السجل \$S1، وللمصفوفة A هو محتوى السجل \$S2:

```
lw $t1,0($s1) # load C[0] into A
sll $t1,$t1,4   # shift left by 4 bits of $t1
sw $t1, 0($s2)  # store C[0] << 4 into A[0]
```

المثال (4-9): أ- اكتب التعليمة المستخدمة لعزل البايت 0 (البتات الثمانية الدنيا) من الكلمة المخزنة في السجل \$t0 ما هي التعليمة التي يمكنها تحقيق ذلك؟
ب- اكتب التعليمة المستخدمة لعزل البايت 1 (البتات ذات الترتيب من 8 إلى 15) من الكلمة المخزنة في السجل \$t1 بحيث يبقى البايت المعزول في السجل \$t1 كأدنى بايت فيه (البايت ذي الترتيب 0).

الحل:

أ- `andi $t0,$t0,0xFF`

تسمى القيمة 0xFF في هذه الحالة قناعاً mask حيث استخدمت لعزل بتات معينة من كلمة معطاة (8 بت في هذا المثال) وجعل باقي بتات الكلمة أصفاراً (بينما لجعل بتات معينة واحداث يلزم استخدام تعليمة ori).

ب- `andi $t1,$t1,0xFF00`

`srl $t1,$t1,8`

أو بدلاً عن ذلك يمكن استخدام التعليمتين التاليتين لتحقيق المطلوب:

`sll $t0,$t0,16`

`srl $t0,$t0,24`

المثال (4-10): اكتب التعليمات اللازمة لجمع العددين 146 + مع -82 ووضع الناتج في السجل \$10، علماً أن تمثيل الأعداد ثنائياً بالإتمام الثنائي.

الحل:

```
ori    $7, $0, 146    # +146 → $7
ori    $8, $0, 82     # 82 → $8
nor    $8, $8, $0     # reflect
ori    $9, $0, 1      # +1
addu   $8, $8, $9     # -82 → $8
addu   $10, $7, $8    # (+146) + (-82) = 64 → $10
```

ويمكن اختصار عدد التعليمات للحصول على المطلوب كما يلي:

```
ori    $7, $0, 146
addiu  $10, $7, -82
```

المثال (4-11): أ- وضح عمل البرنامج التالي باختصار، وما هو ناتج التنفيذ المخزن في السجل \$v0؟

```
li $a0, 5
li $a1, 6
li $v0, 0
Loop: bge $zero, $a1, Exit
      add $v0, $v0, $a0
      addi $a1, $a1, -1
      j Loop
```

Exit:

ب- أعد كتابة البرنامج باستخدام تعليمات حقيقية فقط.

الحل:

أ- يقوم البرنامج بحساب ناتج ضرب محتوى السجل \$a0 بمحتوى السجل \$a1 بشرط أن يكون محتوى السجل \$a1 موجباً، وتخزين الناتج في \$v0، والناتج في هذه الحالة هو \$v0=30.

ب- إعادة كتابة البرنامج باستخدام تعليمات حقيقية:

```
ori $a0,$zero, 5
ori $a1, $zero, 6
ori $v0,$0,$0
Loop: slt $t0,$0,$a1
      beq $t0,$0,Exit
      add $v0,$v0,$a0
      addi $a1,$a1,-1
      j Loop
```

Exit:

المثال (4-12): بفرض أن السجل \$t0 يحوي العنوان القاعدي لمصفوفة مؤلفة من 3 كلمات، اكتب برنامجاً بلغة التجميع MIPS يرجع مجموع عناصر المصفوفة الثلاث في السجل \$t1.

الحل:

start :

```
lw $t2, 0($t0)
lw $t3, 4($t0)
add $t1, $t2, $t3
lw $t2, 8($t0)
add $t1, $t1, $t2
```

طريقة أخرى للحل: بدلاً من تغيير قيمة الانزياح offset يتم تغيير قيمة العنوان القاعدي \$t0:

start :

```
lw $t2, 0($t0)
addi $t0, $t0, 4
lw $t3, 0($t0)
add $t1, $t2, $t3
addi $t0, $t0, 4
lw $t2, 0($t0)
add $t1, $t1, $t2
```

المثال (4-13): بفرض أن السجل \$t0 يحوي العنوان القاعدي لمصفوفة مؤلفة من 3 أعداد صحيحة، اكتب برنامجاً بلغة التجميع MIPS يُرجع في السجل \$t1 القسم الصحيح من المتوسط الحسابي لعناصر المصفوفة الثلاث.

الحل:

start :

```
lw $t2, 0($t0)
```

```

lw $t3, 4($t0)
add $t1, $t2, $t3
lw $t2, 8($t0)
add $t1, $t1, $t2
li $t4, 3
div $t1, $t4
mflo $t1

```

المثال (4-14): اكتب برنامجاً بلغة التجميع MIPS لتنفيذ إجرائية swap التالية المكتوبة بلغة C:

```

void swap(int v[], int k)
{
    int temp; {
        temp = v[k]
        v[k] = v[k+1];
        v[k+1] = temp;
    }
}

```

الحل:

بفرض أن العنوان القاعدي للمصفوفة هو \$a0، وأن k يقابل السجل \$a1:

swap:

```

sll $t0,$a1,2      # $t0=k*4
add $t0,$t0,$a0    # $t0=v + k*4
lw  $t1,0($t0)     # $t1=v[k]

```

```
lw $t2,4($t0)      # $t2=v[k+1]
sw $t2,0($t0)      # v[k]=$t2
sw $t1,4($t0)      # v[k+1]=$t1
jr $ra              # return
```

المثال (4-15): اكتب برنامجاً بلغة التجميع MIPS لتنفيذ العبارة الشرطية التالية:

if ($i \leq j$) $x = x+1$; $z = 1$; else $y = y-1$; $z = 2*z$

الحل:

بفرض أن المتحول i يقابل السجل $$s1$ ، والمتحول z يقابل السجل $$s2$ ، وأن x يقابل $$t1$ ، y يقابل $$t2$ ، z يقابل $$t3$:

```
slt $t0,$s2,$s1      # j<i? (inverse condition)
bne $t0,$zero,else   # if j<i goto else part
addi $t1,$t1,1        # x = x+1
addi $t3,$zero,1      # z = 1
j endif               # skip the else part
else: addi $t2,$t2,-1  # y = y-1
      add $t3,$t3,$t3  # z = z+z
endif:
```

المثال (4-16): اكتب برنامجاً بلغة التجميع MIPS يقوم بنسخ سلسلة المحارف المخزنة في الذاكرة بدءاً من العنوان القاعدي في السجل $$s2$ إلى موقع الذاكرة ذي العنوان القاعدي في السجل $$s1$ ، علماً أن سلسلة المحارف تنتهي بالصففر.

الحل:

```
Loop: lb $t0,0($s2)
      sb $t0,0($s1)
      addi $s2,$s2,1
      addi $s1,$s1,1
      bne $t0,$zero,Loop
```

ملاحظة: إذا كان عدد عناصر السلسلة محدداً وصغيراً فيمكن تغيير قيمة الانزياح offset عند تحميل وتخزين المحارف بدلاً من استخدام الحلقات، كما في التعليمات التالية من أجل سلسلة مؤلفة من 3 محارف:

```
lb $t0,0($s2)
sb $t0,0($s1)
lb $t0,1($s2)
sb $t0,1($s1)
lb $t0,2($s2)
sb $t0,2($s1)
```

المثال (4-17): اكتب برنامجاً بلغة التجميع MIPS يقوم بنسخ عناصر مصفوفة أعداد صحيحة مخزنة في الذاكرة بدءاً من العنوان القاعدي \$s2 إلى موقع الذاكرة ذي العنوان القاعدي \$s1، علماً أن المصفوفة تنتهي بالعدد صفر.

الحل:

```
Loop: lw $t0,0($s2)
      sw $t0,0($s1)
```



```

addi $s2,$s2,4
addi $s1,$s1,4
bne $t0,$zero,Loop

```

ملاحظة: إذا كان عدد عناصر المصفوفة محدداً وصغيراً فيمكن أن يتم تغيير قيمة الانزياح offset في تعليمات التحميل والتخزين بدلاً من استخدام الحلقات:

```

lw $t0,0($s2)
sw $t0,0($s1)
lw $t0,4($s2)
sw $t0,4($s1)
lw $t0,8($s2)
sw $t0,8($s1)

```

المثال (4-18): اكتب برنامجاً بلغة التجميع MIPS ينفذ الحلقة التالية، علماً أن s سلسلة من المحارف char:

```

for (i=0;i<10;i++) {
s[i] = s[i+1];
}

```

الحل:

بفرض أن العنوان القاعدي للسلسلة مخزن في السجل \$s1:

```

addi $s0,$zero,0      # i=0
L1:  slti $t1,$s0,10    # i<10?
      beq $t1,$zero,L2  # no; jump

```

```

add $t2,$s0,$s1      # t2 is address of a[i]
addi $t3,$t2,1        # t3 is addr of a[i+1]
lb $t3,0($t3)         # t3 is a[i+1]
sb $t3,0($t2)         # a[i] = a[i+1]
addi $s0,$s0,1        # i++
j L1                  # go to L1

```

L2:

المثال (4-19): بفرض لدينا البرنامج التالي بلغة التجميع MIPS، والقيم الابتدائية للسجلين \$s2=0, \$t1=10. ما هي القيمة النهائية للسجل \$s2؟

```

LOOP: slt $t2,$0,$t1
      beq $t2, $0, DONE
      addi $t1,$t1,-1
      addi $s2,$s2,2
      j LOOP

```

DONE:

الحل: يقوم البرنامج بمقارنة محتوى السجل \$t1 مع الصفر، فإن كان محتوى \$t1 عدداً موجباً ينقص من \$t1 بمقدار 1، ويزيد على \$s2 بمقدار 2، وتكرر الحلقة طالما $t1 > 0$ ، وبما أن القيم الابتدائية هي $t1=10, s2=0$ ، ففي التكرار الأول للحلقة يصبح $t1=9, s1=2$ ، وتكرر الحلقة 10 مرات إلى أن يصبح $t1=0$ ، وبالتالي فإن محتوى \$s2 النهائي هو: $s2 = 2 \times 10 = 20$.

المثال (4-20): وضّح عمل البرنامج التالي المكتوب بلغة التجميع MIPS.

```

addi $s2,$0,0
addi $t1,$0,0
LOOP: lw $s1,0($s0)
      add $s2,$s2,$s1
      addi $s0,$s0,4
      addi $t1,$t1,1
      slti $t2,$t1,100
      bne $t2,$0,LOOP
DONE:

```

الحل:

نكتب التعليقات الموضحة لعمل التعليمات:

```

addi $s2,$0,0    #$s2=0
addi $t1,$0,0    # $t1 = 0
LOOP: lw $s1,0($s0)  #$s1=Mem[$s0]
      add $s2,$s2,$s1  #$s2=$s2+Mem[$s0]
      addi $s0,$s0,4   # $s0=$s0+4
      addi $t1,$t1,1   #$t1=$t1+1
      slti $t2,$t1,100 # $t2=1 if $t1 < 100; $t2=0 otherwise
      bne $t2,$0,LOOP # branch to LOOP if $t2≠0 ($t1<100)
DONE:

```

يقوم البرنامج بحساب ناتج جمع أول 100 عدد في الذاكرة بدءاً من العنوان

\$S0 وتخزين الناتج في \$S2.

المثال (4-21): اكتب برنامجاً بلغة التجميع MIPS باستخدام تعليمات حقيقية فقط، يقوم بما يلي:

```
switch (k) {  
    case 42      : f=i+j ; break;  
    case 271828  : f=j+g ; break;  
    default     : f=i+g ; break;  
}
```

علماً أن k هو العنصر ذو الترتيب 100 في مصفوفة أعداد صحيحة مخزنة في الذاكرة بدءاً من العنوان القاعدي $a0$.

الحل:

يمكن التعبير عن البرنامج المطلوب باستخدام عبارات شرطية كما يلي:

```
if (k==42)    f = i + j;  
else if (k==271828) f = j+g;  
else f= i + g;
```

ولنفرض أن مقابلة المتحولات المذكورة مع السجلات بالشكل التالي:

$f = \$v0$, $i = \$t0$, $j = \$t1$, $g = \$t2$

علماً أن العدد العشري 271828_{10} يقابل بالتمثيل الست عشري القيمة $0x000425D4$ أي يُمثل ثنائياً على 19 بت، ولكن حقل `imm` في التعليمات من النمط `I_Type` مؤلف من 16 بت فقط، لذا يلزم استخدام التعليمتين `lui` و `ori` لشحن هذا العدد في سجل، كما هو موضح في أسطر البرنامج التالي بلغة التجميع MIPS:

Begin:

```
addi  $s0, $zero, 42    # 42 → $s0
lui   $s1, 0x4
ori   $s1, 0x25D4       # 0x425D4 → $s1
addi  $t6, $0, 400      # t6 is offset (100x4 byte= 400)
add   $t7, $a0, $t6     # t7 is the correct address
lw    $t9, 0($t7)       # t9 is a0[100]
beq   $t9, $s0, Case42  # If (k==42) goto Case42
beq   $t9, $s1, CaseBig # if (k==271828) goto CaseBig
add   $v0, $t0, $t2     # else f= i + g
j     End
```

Case42:

```
add   $v0, $t0, $t1
j     End
```

CaseBig:

```
add   $v0, $t1, $t2
```

End:

المثال (4-22): اكتب برنامجاً بلغة التجميع MIPS يقوم بإيجاد العدد الأكبر في مصفوفة أعداد صحيحة A مخزنة في الذاكرة بدءاً من العنوان القاعدي \$s1، وطول المصفوفة في السجل \$s2، حيث يقوم بنسخ العدد الأكبر إلى السجل \$t0.

الحل:

```

lw    $t0,0($s1)      # initialize maximum to A[0]
addi   $t1,$zero,0     # initialize index i to 0
loop:  addi  $t1,$t1,1   # i++
       beq   $t1,$s2,done # if all elements examined, quit
       sll   $t2,$t1,2   # 4*i → $t2
       add   $t2,$t2,$s1  # address of A[i] → $t2
       lw    $t3,0($t2)  # A[i] → $t3
       slt   $t4,$t0,$t3 # maximum < A[i]?
       beq   $t4,$zero,loop # if not, repeat with no change
       addi  $t0,$t3,0   # if so, A[i] is the new maximum
       j     loop        # repeat
done:

```

المثال (4-23): حوّل كلاً من التعليمات التالية بلغة التجميع MIPS إلى ما يقابلها بلغة الآلة (التمثيل الثنائي للتعليمات)، ثم اكتب قيمها بالتمثيل الست عشري. يمكن الاستعانة بالجدول الواردة في نهاية هذا الفصل.

- 1) add \$s0, \$s1, \$s2
- 2) sub \$t0, \$t3, \$t5
- 3) addi \$t0, \$s3, -12
- 4) lw \$t2, 32(\$0)

الحل:

1) add \$s0, \$s1, \$s2

التعليمة من النمط R_Type وتتألف من الحقول التالية:

OP (6 bit)	Rs (5)	Rt (5)	Rd (5)	S.A (5)	Funct (6)
------------	--------	--------	--------	---------	-----------

op= 000000 (R_Type)

Funct= 32= 100000 (Function: add)

Rs= \$s1= 17 = 10001

Rt= \$s2= 18 = 10010

Rd= \$s0= 16= 10000

S.A (Shift Amount) = 00000

وبالتالي فإن هذه التعليمة بلغة الآلة هي:

000000 10001 10010 10000 00000 100000
= 0x02328020

2) sub \$t0, \$t3, \$t5

التعليمة من النمط R_Type وفيها:

op= 000000 (R_Type)

Funct= 34= 100010 (Function: sub)

Rs= \$t3= 11 = 01011

Rt= \$t5= 13 = 01101

Rd= \$t0= 8= 01000

S.A (Shift Amount) = 00000

وبالتالي فإن هذه التعليمة بلغة الآلة هي:

000000 01011 01101 01000 00000 100010

= 0x016D4022

3) addi \$t0, \$s3, -12

التعليمة من النمط I_Type وتتألف من الحقول التالية:

OP (6 bit)	Rs (5)	Rt (5)	Immediate (16 bit)
------------	--------	--------	--------------------

op= 8 = 001000 (addi)

Rs= \$s3= 19 = 10011

Rt= \$t0= 8= 01000

Imm= -12= 1111 1111 1111 0100

وبالتالي فإن هذه التعليمة بلغة الآلة هي:

001000 10011 01000 1111 1111 1111 0100 = 0x2268FFF4

4) lw \$t2, 32(\$0)

التعليمة من النمط I_Type وفيها:

op= 35 = 100011 (lw)

Rs= \$0= 00000

Rt= \$t2= 10= 01010

Imm= 32= 0000 0000 0010 0000

وبالتالي فإن هذه التعليمة بلغة الآلة هي:

100011 00000 01010 0000 0000 0010 0000= 0x8C0A0020

المثال (4-24): حوّل كلاً من التعليمات التالية بلغة التجميع MIPS إلى ما يقابلها بلغة الآلة، ثم اكتب قيمها بالتمثيل الست عشري، يمكن الاستعانة بالجدول الواردة في نهاية هذا الفصل.

- 1) sll \$t0, \$s1, 4
- 2) jr \$t0
- 3) mult \$t0, \$t1
- 4) mflo \$t2
- 5) mfhi \$t3

الحل:

التعليمات السابقة هي من النمط R_type، ولكنها لا تحتاج كامل المعاملات الثلاثة، وإنما تستخدم السجلات حسب طبيعة التعليمات، ويوضع عند الباقي أصفار:

1) sll \$t0, \$s1, 4 (sll Rd, Rt; Rd = Rt<<4)

التعليمات من النمط R_Type، وفيها حقل مقدار الإزاحة Shift Amount يساوي 4، أي أنه تتم الإزاحة بمقدار 4 بت:

قيم الحقول عشرياً: Op=0, rs=0, rt=17, rd=8, S.A=4, Funct=0

وثنائياً: 000000 00000 10001 01000 00100 000000

وبالنظام الست عشري: 0x00114100

2) jr \$t0 # (jr Rs ; Rs= \$t0=\$8)

قيم الحقول عشرياً: Op=0, rs=8, rt=0, rd=0, S.A=0, Funct=8

وثنائياً: 000000 01000 00000 00000 00000 001000

وبالنظام الست عشري: 0x01000008

3) mult \$t0, \$t1 # (mult Rs, Rt)

قيم الحقول عشرياً: Op=0, rs=8, rt=9, rd=0, S.A=0, Funct=24

وثنائياً: 000000 01000 01001 00000 00000 011000

وبالنظام الست عشري: 0x01090018

4) mflo \$t2 # (mflo Rd)

قيم الحقول عشرياً: Op=0, rs=0, rt=0, rd=10, S.A=0, Funct=18

وثنائياً: 000000 00000 00000 01010 00000 010010

وبالنظام الست عشري: 0x00005012

5) mfhi \$t3 # (mfhi Rd)

قيم الحقول عشرياً: Op=0, rs=0, rt=0, rd=11, S.A=0, Funct=16

وثنائياً: 000000 00000 00000 01011 00000 010000

وبالنظام الست عشري: 0x00005810.

المثال (4-25): مثل التعليمات التالية بلغة الآلة، ثم اكتبها بالتمثيل الست

عشري. lui \$t6, 0xC # 0x000C0000 → \$t6

الحل:

التعليمات lui من النمط I_Type وفيها حقل رمز التعليمات op=15، والحقول:

Rs=0, Rt=\$t6=14, imm=0xC وبالتالي فالتمثيل الثنائي لها:

001111 00000 01110 0000 0000 0000 1100

وبالنظام الست عشري: 0x3C0E000C.

المثال (4-26): اكتب تعليمات bne الواردة في البرنامج التالي بلغة الآلة، ثم

اكتبها بالتمثيل الست عشري.

```

loop: add $t1, $a0, $s0 # ← (loop)
        lb $t1, 0($t1)
        add $t2, $a1, $s0
        sb $t1, 0($t2)
        addi $s0, $s0, 1
        bne $t1, $0, loop # (PC)
        lw $s0, 0($sp) # ← (PC+4)
loop2: # (loop2)

```

الحل:

تعليلة bne من النمط I_Type وفيها 5 ، op= ، Rs=\$t1=\$9 ، Rt=\$0 ، أما الحقل imm فيتم حسابه كما يلي:

عدد التعليلات الفاصلة بين PC+4 وبين عنوان التفرع loop هو 6 تعليلات، وبما أن التفرع يحصل للأعلى فيكون مقدار imm سالباً أي -6 ، ونمثله ثنائياً بالإتمام الثنائي كما يلي:

6= 0000 0000 0000 0110
-6= 1111 1111 1111 1010

قيم حقول التعليلة bne ثنائياً:

000101 01001 00000 1111 1111 1111 1010

وبالنظام الست عشري: 0x1520FFFA.

ملاحظة: لو كانت تعليلة التفرع السابقة هي bne \$t1, \$0, loop2 فعندئذ تكون القيمة العددية في الحقل imm هي +1 (يساوي عدد التعليلات الفاصلة بين PC+4 وبين عنوان التفرع loop، وهو عدد موجب لأن التفرع يحصل إلى الأسفل).

علماً أن عنوان وجهة التفرع يتم حسابه كما يلي:

$$\text{Branch Target Address} = (\text{PC}+4) + [\text{S.Ext}(\text{imm16}) \times 4].$$

المثال (4-27): مثل تعليمة jal التالية بلغة الآلة، ثم بالتمثيل الست عشري.

```
0x0040005C      jal sum
...
0x004000A0      sum: add $v0, $a0, $a1
```

الحل:

تعليمة jal هي من النمط J_Type وفيها حقلان: حقل رمز التعليمة op=3، وحقل عنوان وجهة القفز target address مؤلف من 26 بت، ويحسب كما يلي:

عنوان وجهة القفز (كاملاً) = 0x004000A0

التمثيل الثنائي لعنوان وجهة القفز:

0000 0000 0100 0000 0000 0000 1010 0000

علماً أنه يتم تشكيل عنوان القفز كما يلي:

PC [31..28] || target address (26bits) || 00

وبالتالي للحصول على حقل عنوان الوجهة المؤلف من 26 بت يتم حذف البتات الأربعة الأعلى وزناً، وحذف أدنى بتين من العنوان كاملاً، فنحصل على:

00 0001 0000 0000 0000 0010 1000

فيكون تمثيل التعليمة ثنائياً (بلغة الآلة) بالشكل:

000011 00 0001 0000 0000 0000 0010 1000

وبالتمثيل الست عشري: 0x0C100028.

000000 000100100000000000000011000 (R_Type)

000000 000000000000001000000010010 (R_Type)

001000 010000100000000000 00000001 (I_Type)

000010 000001000000000000000000010 (j_Type)

وبعد معرفة نمط كل تعليمة يتم توزيع البتات على حقول كل من الأنماط الثلاثة

كما يلي:

Op=001000, rs=00000, rt= 01000, imm=0000 0000 0000 0001

Op=001000, rs=00000, rt=00010, imm=0000 0000 0000 0001

Op=000000,rs=00100,rt=01000,rd=01001,SA=00000,F=101010

Op=000101,rs=01001, rt=00000, imm= 0000 0000 0000 0100

Op=000000,rs=00010,rt=01000,rd=00000,SA=00000,F=011000

Op=000000,rs=00000,rt=00000,rd=00010,SA=00000,F=010010

Op=001000,rs=01000, rt= 01000, imm= 0000 0000 0000 0001

Op=000010, address target= 000001000000000000000000010

ومن ثم يتم كتابة التعليمات بلغة التجميع MIPS بالاستعانة بالجدول الواردة في

نهاية هذا الفصل:

addi \$t0, \$zero, 1

addi \$v0, \$zero, 1

slt \$t1, \$a0, \$t0

bne \$t1, \$0, L1

mult \$v0, \$t0

mflo \$v0

addi \$t0, \$t0, 1

j L2

يلزم معرفة عنوان التسميات L1, L2، علماً أن البرنامج يبدأ عند العنوان
:0x00400000

بالعودة إلى التمثيل الثنائي للتعليمية bne نجد أن محتوى الحقل $imm=+4$ ، أي
أن العنوان L1 يبعد عن $PC+4$ بمقدار 4 تعليمات للأسفل، وبالتالي فإن العنوان L1
يتوضع في نهاية البرنامج بعد تعليمية القفز.

وبالعودة إلى التمثيل الثنائي للتعليمية نجد أن الحقل address target مؤلف
من 26 بت، وللحصول على العنوان كاملاً يتم إضافة 4 بت العليا من السجل PC
لتصبح أعلى بتات في العنوان، مع إضافة 00 إلى أدنى العنوان، فيتشكل عنوان القفز
كما يلي:

PC[31..28] || target address (26bits) || 00

ويكون التمثيل الثنائي للعنوان كاملاً:

0000 0000010000000000000000000010 00

وبالتحويل إلى الترميز الست عشري نحصل على عنوان وجهة القفز:
0x00400008

ولدينا عناوين تعليمات البرنامج كما يلي:

0x00400000

0x00400004

0x00400008

0x0040000C

0x00400010

0x00400014

0x00400018

0x0040001C

وبالتالي فإن عنوان وجهة القفز L2 يتوضع عند التعليمة الثالثة.

نعيد كتابة التعليمات مع عناوين التفرع والقفز ومع التعليقات لتوضح عمل

البرنامج:

```
addi $t0, $zero, 1    # i = 1
addi $v0, $zero, 1    # v = 1
L2:  slt  $t1, $a0, $t0  # if ( arg < i )
     bne $t1, $0, L1    # Exit if (arg < i ) { bgt $t0, $a0, L1}
     mult $v0, $t0
     mflo $v0           # v *= i
     addi $t0, $t0, 1   # i ++
     j    L2           # loop
```

L1:

عمل البرنامج: إيجاد العامل لمحتوى السجل \$a0، وحفظ الناتج في السجل

\$v0، ويقابله بلغة C:

```
int v = 1;
for (int i = 1 ; i < arg ; i ++ ) {
    v = v * i;
}
```


المثال (4-29): اكتب برنامجاً بلغة التجميع MIPS يقوم باستدعاء تابع لحساب قوة عدد x^y وإرجاع الناتج إلى البرنامج الرئيسي main.

الحل:

لاستدعاء أي تابع فإننا نستخدم التعليمة jal، وفي نهاية التابع نضع التعليمة jr \$ra للعودة إلى البرنامج المستدعي.

main:

```
li $a0,5    # pass arg's to function
```

```
li $a1,3
```

```
jal power
```

endmain:

```
#--- Function to find and return  $x^y$  ---#
```

power:

```
li $v0,1
```

```
li $t0,0
```

powLoop:

```
mul $v0,$v0,$a0
```

```
addi $t0,$t0,1
```

```
blt $t0,$a1,powLoop
```

```
jr $ra # return to caller
```

عند تنفيذ البرنامج يعطي الناتج $5^3 = 125$.

المثال (4-30): اكتب إجرائية بلغة التجميع MIPS تكافئ الإجرائية التالية:

```

int proc1 (int g, h, i, j)
{ int f;
  f = (g + h) - (i + j);
  return f;
}

```

إذا علمت أن المتحولات g, h, i, j تقابل السجلات $\$a0, \$a1, \$a2, \$a3$ على الترتيب، وأن المتحول f يقابل السجل $\$s0$ ، ويتم إرجاع الناتج في السجل $\$v0$.
الحل:

بما أننا نستخدم السجل $\$s0$ ضمن الإجرائية وهو من السجلات المحفوظة **saved registers** وبالتالي يجب حفظه في المكس **stack** قبل التعديل على محتواه، ويتم التعامل مع المكس باستخدام سجل مؤشر المكس كما هو موضح في البرنامج التالي:

```

proc1:
    addi $sp, $sp, -4      # Make room for one word
    sw   $s0, 0($sp)      # Save $s0 on stack
    add  $t0, $a0, $a1     # g+h
    add  $t1, $a2, $a3     # i+j
    sub  $s0, $t0, $t1     # f= (g+h) - (i+j)
    add  $v0, $s0, $zero   # Result
    lw   $s0, 0($sp)      # Restore $s0
    addi $sp, $sp, 4       # Adjust stack pointer
    jr   $ra              #Return to caller

```

ملاحظة: عند استخدام أحد السجلات المحفوظة \$s0..\$s7 ضمن الإجرائية أو التابع يلزم حفظها في المكس قبل التعديل عليها، واسترجاعها قبل العودة إلى البرنامج المستدعي.

المثال (4-31): اكتب برنامجاً بلغة التجميع MIPS يحسب قيمة العدد رقم n من سلسلة فيبوناتشي بشكل عودي.

الحل: بفرض أن الرقم n يقابل السجل \$a0:

fib:

addi \$sp, \$sp, -12

sw \$ra, 8(\$sp)

sw \$s0, 4(\$sp)

addi \$v0, \$zero, 1

beq \$a0, \$zero, fin

addi \$t0, \$zero, 1

beq \$a0, \$t0, fin

addi \$a0, \$a0, -1

sw \$a0, 0(\$sp)

jal fib

lw \$a0, 0(\$sp)

addi \$a0, \$a0, -1

add \$s0, \$v0, \$zero

jal fib

add \$v0, \$v0, \$s0

fin:

lw \$s0, 4(\$sp)

lw \$ra, 8(\$sp)

addi \$sp, \$sp, 12

jr \$ra

عند تنفيذ البرنامج من أجل 5 $a0 =$ يرجع النتيجة = 8.



تمارين غير محلولة:

التمرين (4-1): ما هو ناتج كل من التعليمات التالية المخزن في سجل الوجهة

للتعليمة؟

```
addi $3, $0, 1    # $3 = ?  
sll  $10, $3, 1    # $10 = ?  
sll  $11, $3, 2    # $11 = ?  
addi $3, $0, 15    # $3 = ?  
srl  $15, $3, 1    # $15 = ?  
srl  $16, $3, 2    # $16 = ?
```

التمرين (4-2): اكتب برنامجاً بلغة التجميع MIPS يقوم بنسخ البت الأدنى ذي الترتيب 0 إلى البت الأعلى ذي الترتيب 31 في السجل \$8 (مثلاً إذا كان المحتوى الابتدائي للسجل $0x7fff0000$ فالمحتوى النهائي له بعد تنفيذ البرنامج هو $0xffff0000$).

التمرين (4-3): أعد كتابة البرنامج التالي باستخدام تعليمات حقيقية فقط:

```
li    $v0,0  
Loop: bge $zero,$a1,Exit  
      add $v0,$v0,$a0  
      sub $a1,$a1,1  
      j   Loop  
Exit:
```

التمرين (4-4): ما هو عمل الجزء التالي من البرنامج المكتوب بلغة التجميع

.MIPS

```
lui $s0, 0x1000
ori $s0, $s0, 0x7000
lw $t1, 0($s0)
sll $t1, $t1, 3
sw $t1, 0($s0)
lw $t1, 4($s0)
sll $t1, $t1, 3
sw $t1, 4($s0)
```

التمرين (5-4): ما هو عمل البرنامج التالي المكتوب بلغة التجميع .MIPS

```
test: slt $s1, $t2, $zero
      beq $s1, $zero, L1
      sub $t2, $zero, $t2
L1:   slt $s1, $t3, $zero
      beq $s1, $zero, L2
      sub $t3, $zero, $t3
L2:   add $t1, $t2, $t3
      jr $ra
```

التمرين (6-4): أ- ما هو عمل البرنامج التالي المكتوب بلغة التجميع .MIPS

```

begin: addi $t0, $zero, 0
      addi $t1, $zero, 1
loop:  slt  $t2, $a0, $t1
      bne  $t2, $zero, finish
      add  $t0, $t0, $t1
      addi $t1, $t1, 2
      j    loop
finish: add $v0, $t0, $zero

```

ب- ما هو الناتج من أجل $a0 = 10$.

ج- مثل التعليمات السابقة ثنائياً إذا علمت أن بداية البرنامج begin عند العنوان 0x00400000.

التمرين (4-7): اكتب اجرائية StrLen بلغة MIPS تأخذ معاملاً واحداً وهو السجل \$a0 الذي يدل على بداية سلسلة المحارف، ونستدل على نهايتها بوجود null، تقوم الاجرائية بإيجاد طول سلسلة المحارف وترجع هذا الطول في السجل \$v0 (مثلاً إن كانت سلسلة المحارف هي hello فالقيمة المرجعة هي 5).

التمرين (4-8): بفرض لدينا التابع التالي المكتوب بلغة التجميع MIPS:

```

func:
      li    $v0, 0
      li    $t0, 0
L1:   add   $t1, $a0, $t0
      lb    $t2, 0($t1)

```

```

    beq    $t2, $zero, L3
    bne    $t2, $a1, L2
    add    $v0, $v0, 1
L2:  add    $t0, $t0, 1
    j      L1
L3:  jr     $ra

```

اكتب البرنامج السابق بلغة عالية المستوى، ثم اشرح عمله باختصار.

التمرين (4-9): بفرض لدينا البرنامج التالي بلغة الآلة تم تمثيل تعليماته بالنظام الست عشري كما يلي:

```

0x8d0b0000
0x8d0c0004
0x016c5020
0xad0a0008
0x21080004
0x2129ffff
0x0009682a
0x15a0fff8

```

المطلوب كتابة التعليمات المقابلة لها بلغة التجميع MIPS، علماً أن البرنامج يبدأ عند العنوان 0x00400000، ثم وضح عمل البرنامج باختصار.

الجدول التالي يوضح أرقام وأسماء السجلات في معالج MIPS وملخصاً عن استخدامها في البرامج، يليه جداول بأهم التعليمات في معالج MIPS والقيم العددية لحقول Op_code، Func فيها:

اسم السجل	رقمه	استخدام السجل
zero	0	يحتوي دوماً القيمة صفر
at	1	محجوز للمجمّع، ويستخدم في التعليمات الزائفة
v0- v1	2-3	ناتج التعبيرات expression أو التتابع function
a0,a1,a2,a3	4,5,6,7	أربعة معاملات Arguments
t0- t7	8-15	سجلات مؤقتة Temporary لا تُحفظ
t8, t9	24,25	عند الاستدعاء
s0-s7	16-23	سجلات محفوظة Saved تُحفظ عند الاستدعاء
k0,k1	26,27	محجوزة لنواة نظام التشغيل OS kernel
gp	28	مؤشر شمولي Global Pointer
sp	29	مؤشر المكس Stack
fp	30	مؤشر الإطار Frame
ra	31	عنوان العودة Return address

قيم الحقل Func للتعليمات من النمط R_Type (op_code=0)				
الرمز	اسم التعليمة	عمل التعليمة	Func(16)	Func(10)
add	Add	$R[rd] = R[rs] + R[rt]$	20	32
addu	Add Unsigned	$R[rd] = R[rs] + R[rt]$	21	33
and	And	$R[rd] = R[rs] \& R[rt]$	24	36
jr	Jump Register	$PC=R[rs]$	8	8
nor	Nor	$R[rd] = \sim (R[rs] \mid R[rt])$	27	39
xor	xor	$R[rd] = R[rs] \mid R[rt]$	26	38
or	Or	$R[rd] = R[rs] \mid R[rt]$	25	37
slt	Set Less Than	$R[rd] = (R[rs] < R[rt])? 1:0$	2a	42
sltu	Set Less Than Unsign	$R[rd] = (R[rs] < R[rt])? 1:0$	2b	43
sll	Shift Left Logical	$R[rd] = R[rt] \ll \text{shamt}$	0	0
srl	Shift Right Logical	$R[rd] = R[rt] \gg \text{shamt}$	2	2
sra	Shift Right Arith	$R[rd] = R[rt] \ggg \text{shamt}$	3	3
sub	Subtract	$R[rd] = R[rs] - R[rt]$	22	34
subu	Subtract Unsigned	$R[rd] = R[rs] - R[rt]$	23	35
div	Divide	$Lo=R[rs]/R[rt];Hi=R[rs]\%R[rt]$	1a	26
divu	Divide Unsigned	$Lo=R[rs]/R[rt];Hi=R[rs]\%R[rt]$	1b	27
mfhi	Move From Hi	$R[rd] = Hi$	10	16
mflo	Move From Lo	$R[rd] = Lo$	12	18
mult	Multiply	$\{Hi,Lo\} = R[rs] * R[rt]$	18	24
multu	Multiply Unsigned	$\{Hi,Lo\} = R[rs] * R[rt]$	19	25

قيم حقل Op_code للتعليمات من النمط I_Type				
الرمز	اسم التعليمة	عمل التعليمة	Op_code 16	Op_code 10
beq	Branch On Equal	if(R[rs]==R[rt]) PC=PC+4+BranchAddr	4	4
bne	Branch On Not Equal	if(R[rs]!=R[rt]) PC=PC+4+BranchAddr	5	5
addi	Add Immediate	R[rt] = R[rs] + SignExt	8	8
addiu	Add Imm. Unsigned	R[rt] = R[rs] + SignExt	9	9
slti	Set Less Than Imm.	R[rt] = (R[rs] < SignExtImm)? 1 : 0	a	10
sltiu	Set Less Than Imm. Unsign	R[rt]=(R[rs]< SignExt) ? 1: 0	b	11
andi	And mmediate	R[rt] = R[rs] & ZeroExt	c	12
ori	Or Immediate	R[rt] = R[rs] ZeroExt	d	13
lb	Load Byte	R[rt]=S_Ext[address]7:0	20	32
lbu	Load Byte Unsigned	R[rt]=ZeroExt[address]7:0	24	36
lui	Load Upper Imm.	R[rt] = {imm, 16'b0}	f	15
lw	Load Word	R[rt]=M[R[rs]+SignExtImm]	23	35
sb	Store Byte	M[R[rs]+SignExtImm](7:0)= R[rt](7:0)	28	40
sw	Store Word	M[R[rs]+SignExtImm] = R[rt]	2b	43

قيم حقل Op_Code للتعليمات من النمط J_Type				
الرمز	اسم التعليمة	عمل التعليمة	Op_code(16)	Op_code(10)
j	Jump	PC=JumpAddr	2	2
jal	Jump And Link	R[31]=PC+8; PC=JumpAddr	3	3

بعض تعليمات الفاصلة العائمة FP		
add.s	FP Add Single	$F[fd] = F[fs] + F[ft]$
add.d	FP Add Double	$\{F[fd], F[fd+1]\} = \{F[fs], F[fs+1]\} + \{F[ft], F[ft+1]\}$
div.s	FP Divide Single	$F[fd] = F[fs] / F[ft]$
div.d	FP Divide Double	$\{F[fd], F[fd+1]\} = \{F[fs], F[fs+1]\} / \{F[ft], F[ft+1]\}$
mul.s	FP Multiply Single	$F[fd] = F[fs] * F[ft]$
mul.d	FP Multiply Double	$\{F[fd], F[fd+1]\} = \{F[fs], F[fs+1]\} * \{F[ft], F[ft+1]\}$
sub.s	FP Subtract Single	$F[fd] = F[fs] - F[ft]$
sub.d	FP Subtract Double	$\{F[fd], F[fd+1]\} = \{F[fs], F[fs+1]\} - \{F[ft], F[ft+1]\}$
lwc1	Load word to coprocessor1	$F[rt] = M[R[rs] + \text{SignExtImm}]$
ldc1	Load FP Double	$F[rt] = M[R[rs] + \text{SignExt}];$ $F[rt+1] = M[R[rs] + \text{SignExt} + 4]$
swc1	Store word to FP coprocessor1	$M[R[rs] + \text{SignExtImm}] = F[rt]$
sdc1	Store FP Double	$M[R[rs] + \text{SignExt}] = F[rt];$ $M[R[rs] + \text{S_Ext} + 4] = F[rt+1]$

الفصل الخامس

ممر المعطيات Data Path

مقدمة:

يتناول هذا الفصل بنية ممر المعطيات في معالج MIPS وكيف يتم تنفيذ تعليمات MIPS المختلفة عبر هذا الممر، والذي يتكون بشكل أساسي من ذاكرة لتخزين التعليمات والمعطيات Instruction & Data Memory وملف السجلات Register File، ووحدة الحساب والمنطق ALU، وسجل عداد البرنامج program counter (PC) والذي يحتفظ بعنوان التعليمة التالية التي سيتم تنفيذها، وممدد بالإشارة أو بالأصفار sign/zero extender، مع ما يلزم من خطوط توصيل ونواخب وجوامع، بالإضافة إلى خطوط التحكم. حيث يتم غالباً جلب التعليمة من الذاكرة، ثم تحليلها، ثم تنفيذها وكتابة النتيجة، وبالتالي يتم نقل المعطيات من السجل إلى الذاكرة، أو من الذاكرة إلى السجل، أو من سجل إلى سجل، بعد إجراء العمليات المطلوبة عليها. في ممر المعطيات أحادي الدور Single Cycle Data path نجد أن كل تعليمة تنفذ في دور ساعة واحد، بينما في ممر المعطيات متعدد الأدوار Multi Cycle Data path نجد أن التعليمات تنفذ على عدة مراحل حسب حاجتها، وكل مرحلة تنفذ خلال دور ساعة، وبالتالي يختلف عدد الأدوار حسب نوع التعليمات.

وفيما يلي أمثلة توضح تنفيذ التعليمات الأساسية في لغة التجميع MIPS على ممر معطيات أحادي الدور، ثم أمثلة مشابهة على ممر المعطيات متعدد الأدوار، وبعدها مجموعة من الأمثلة تتعلق بأداء المعالجات في حالات الممرات أحادية الدور أو متعددة الأدوار.

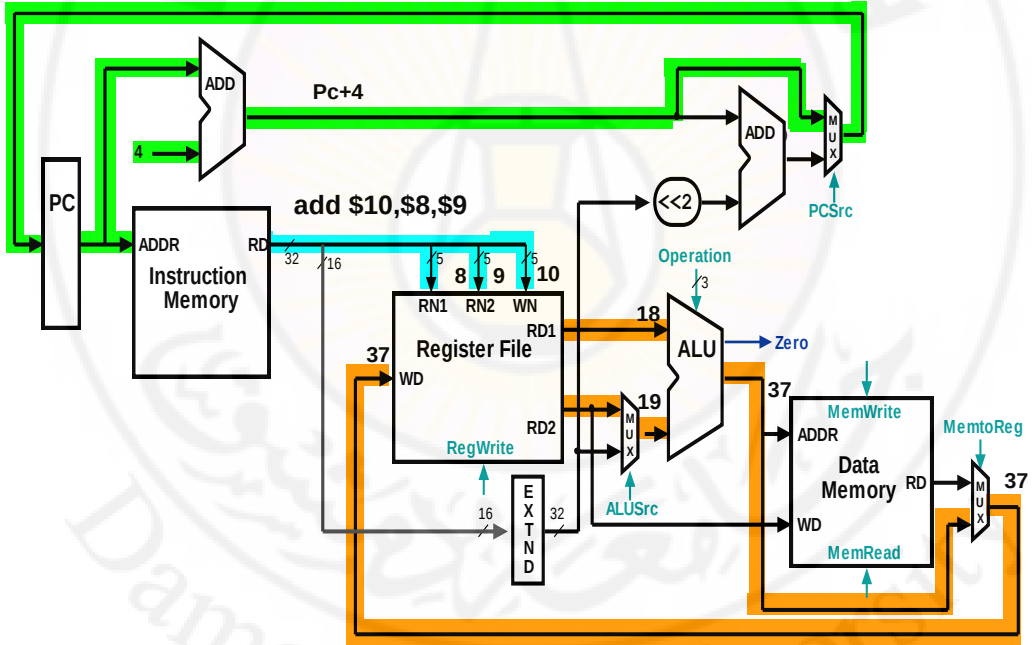
أمثلة محلولة:

المثال (5-1): وضّح مسار المعطيات على معالج MIPS أحادي الدور Single Cycle عند تنفيذ التعليمة `add $10,$8,$9` بفرض أن القيم الابتدائية للسجلات المستخدمة كالتالي: $\$8=18$, $\$9=19$, $\$10=200$.

ما هي القيمة النهائية التي يتم كتابتها؟ وأين تُكتب؟

الحل:

التعليمة المطلوبة من النمط R-type ولها الشكل التالي: `add rd, rs, rt`



يتم كتابة ناتج عملية الجمع (العدد 37) في السجل \$10 ضمن ملف السجلات.

المثال (5-2): وضّح مسار المعطيات على معالج MIPS أحادي الدور عند

تنفيذ كل من التعليمتين $\text{addi } \$9, \$8, -1$ ، $\text{ori } \$9, \$8, 3$ بفرض أن القيم الابتدائية للسجلات المستخدمة في التعليمتين: $\$9=4$, $\$8=6$.

ما هي القيمة النهائية التي يتم كتابتها في كل من الحالتين؟

الحل:

التعليمتان من النمط I-type ، لهما الشكل inst rt, rs, imm حيث أن التمثيل الثنائي للقيمة العددية (-1) في التعليمية addi هو: 1111111111111111 تمدد بالإشارة sign_ext لتحويل العدد من 16 بت إلى 32 بت حتى يتم جمعه مع محتوى السجل $\$8$.

$\text{Sign_ext(Imm)} = 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111$

$\$8 = 6 = 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0110$

ناتج جمعهما هو:

0000 0000 0000 0000 0000 0000 0000 0101

وهو العدد (5) ويخزن هذا الناتج في السجل $\$9$ في ملف السجلات.

أما في التعليمية ori المنطقية فإن القيمة العددية (3) يتم تمديدها بالأصفار zero_ext وليس بالإشارة.

$\$8 = 6 = 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0110$

$\text{zero_ext(Imm)} = 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0011$

بإجراء عملية Or بينهما نحصل على الناتج:

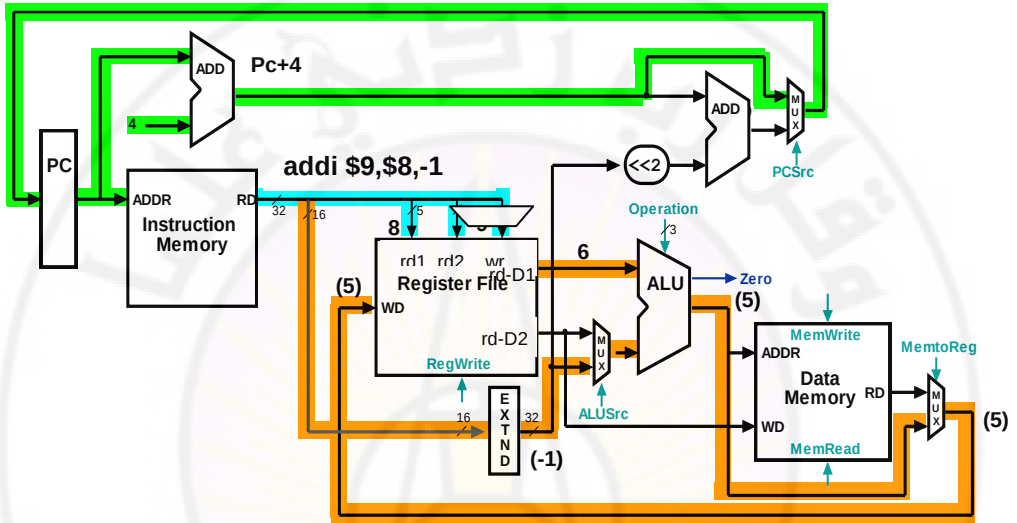
0000 0000 0000 0000 0 0000 0000 0000 0111

ويقابل العدد (7) بالنظام العشري، ويخزن الناتج في السجل $\$9$ في ملف

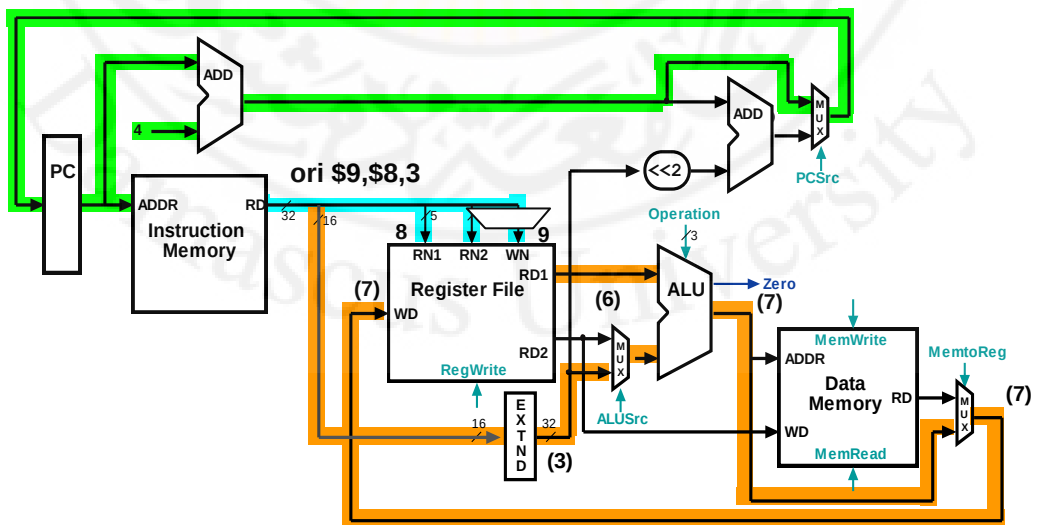
السجلات.

وفيما يلي توضيح لممر المعطيات مع القيم العددية في كل من التعليمتين
السابقتين:

التعليمة : `addi $9,$8,-1`



التعليمة : `ori $9,$8,3`



المثال (3-5): وضّح مسار المعطيات على معالج MIPS أحادي الدور عند تنفيذ التعليمة $lw \$9,4(\$8)$.

بفرض أن القيم الابتدائية للسجلات المستخدمة: $\$9=8$, $\$8=16$ ، وأن محتوى ذاكرة المعطيات الابتدائي أصفار.

ما هي القيمة النهائية التي يتم كتابتها؟ وأين تُكتب؟

الحل:

التعليمة المطلوبة من النمط l -type، ولها الشكل التالي: $lw rt, offset(rs)$ حيث أن القيمة العددية (4) تمثل مقدار الانزياح $offset$ عن العنوان القاعدي $base$ address المخزنة قيمته في السجل $\$8$ ، بحيث ينتج عن جمع هاتين القيمتين عنوان الذاكرة الذي سيتم تحميل محتواه إلى السجل $\$9$ ، مع ضرورة تمديد القيمة العددية $offset$ بالإشارة لتحويل العدد من 16 بت إلى 32 بت حتى يتم جمعها مع محتوى السجل $\$8$.

$rt \leftarrow MEM[rs + sign_ext(imm16)]$

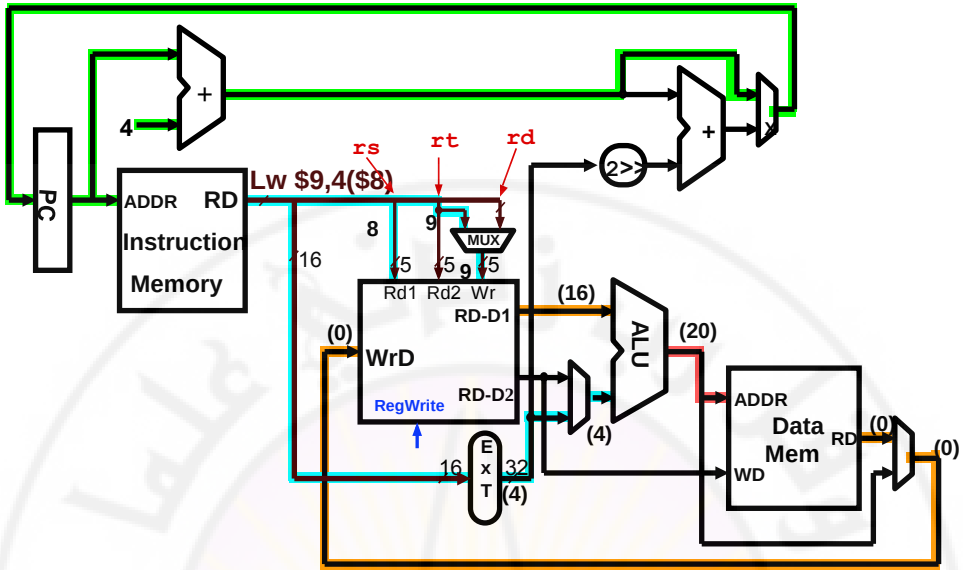
$\$9 \leftarrow MEM[\$8 + sign_ext(4)]$

$\$9 \leftarrow MEM[16 + sign_ext(4)]$

$\$9 \leftarrow MEM[20]$

وبما أن محتوى الذاكرة في أي موقع هو "0"، فإنه يتم تحميل القيمة 0 من الموقع ذي العنوان 20 بالذاكرة إلى السجل $\$9$.

والشكل التالي يوضح ممر المعطيات عند تنفيذ هذه التعليمة:



نلاحظ إضافة الناخب على مدخل الكتابة في ملف السجلات ليحدد السجل الوجهة للتعليلة المنفذة، هل هو rd كما في تعليمات add, sub, or وغيرها من تعليمات النمط R-type، أم هو rt كما في تعليمات addi, ori, lw وغيرها من النمط I-Type.

المثال (4-5): وضّح مسار المعطيات على معالج MIPS أحادي الدور عند تنفيذ التعليلة sw \$10,4(\$8) بفرض أن القيم الابتدائية للسجلات المستخدمة: \$8=16 , \$10=30، وأن محتوى ذاكرة المعطيات الابتدائي أصفار.

ما هي القيمة النهائية التي يتم كتابتها؟ وأين تُكتب؟

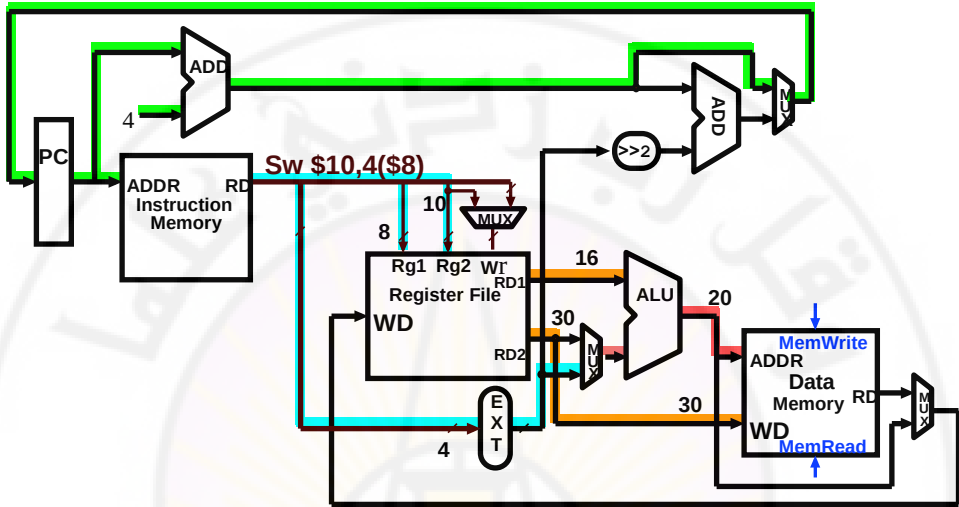
الحل:

التعليلة المطلوبة هي من النمط I-type، وتُكتب بالشكل التالي:

sw rt, offset (rs)

حيث أن القيمة العددية (4) تمثل مقدار الانزياح عن العنوان القاعدي المخزن في السجل \$8 بحيث ينتج عن جمع هاتين القيمتين عنوان الذاكرة الذي سيتم تخزين

محتوى السجل \$10 فيه، مع ضرورة تمديد القيمة العددية offset بالإشارة لتحويل العدد من 16 بت إلى 32 بت حتى يتم جمعها مع محتوى السجل \$8.



$\text{MEM}[20] \leftarrow (\$10=30)$

أي تقوم هذه التعليمة بتخزين القيمة 30 في موقع الذاكرة ذي العنوان 20.

المثال (5-5): وضّح مسار المعطيات على معالج MIPS أحادي الدور عند تنفيذ التعليمة `Beq` الواردة في جزء البرنامج التالي، بفرض أن القيم الابتدائية للسجلات المستخدمة: $\$8=15$, $\$4=15$ ، وأن عناوين التعليمات مكتوبة بالنظام العشري في بداية التعليمات، وضّح عمل التعليمة.

100. `Beq $4, $8, Label`

104. `Add $5, $5, $4`

108. `Add $6, $6, $5`

112. `Label:`

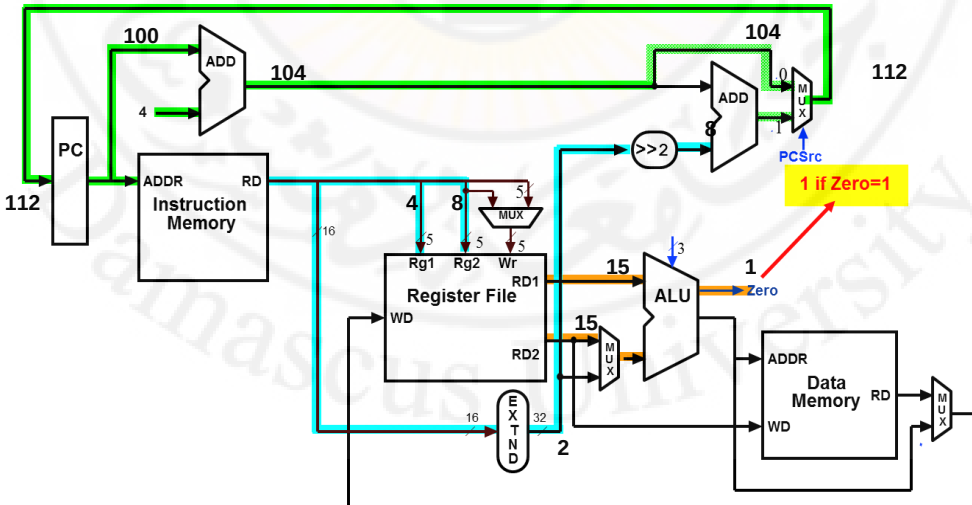
الحل:

نلاحظ أن شرط التفرع للتعليمية Beq المعطاة محقق لتساوي محتوى السجلين \$4, \$8 وبالتالي فإن الخرج zero في ALU يعطي القيمة 1 بسبب تساويهما، وهذا الخرج يدخل إلى خط التحكم بالناخب، فيتم التفرع إلى العنوان Label الذي يبعد عن (PC+4=104) بمقدار تعليمتين، أي أن قيمة محتوى الحقل imm لتعليمية Beq هو 2₁₀، ممثلاً ثنائياً على 16 بت، ويتم تمديده بالإشارة إلى 32 بت، ثم إزاحته لليسر خانتين وهذه الإزاحة تكافئ ضرب imm بالعدد 4، ثم إضافة الناتج (8) إلى (PC+4=104) للحصول على عنوان التفرع، وبالتالي فإن الناخب الموصول إلى PC سيختار في هذه الحالة المدخل (1) الذي يحقق التفرع إلى العنوان Label أي إلى التعليمية ذات العنوان 112.

Beq \$4, \$8 , Label #\$4== \$8 ⇒

PC ← [(PC + 4) + sign_ext(imm16)] || 00⇒

PC ← 112



المثال (5-6): وضّح مسار المعطيات من أجل معالج MIPS أحادي الدور عند تنفيذ التعليمة: `jal Label` إذا علمت أن عنوان هذه التعليمة هو `0x00400000` وأن عنوان وجهة القفز `Label` هو `0x0040000C`.

الحل:

$\text{jal address} \Leftrightarrow \text{j address} \ \& \ (\text{PC}+4) \rightarrow \ra

عنوان تعليمة القفز:

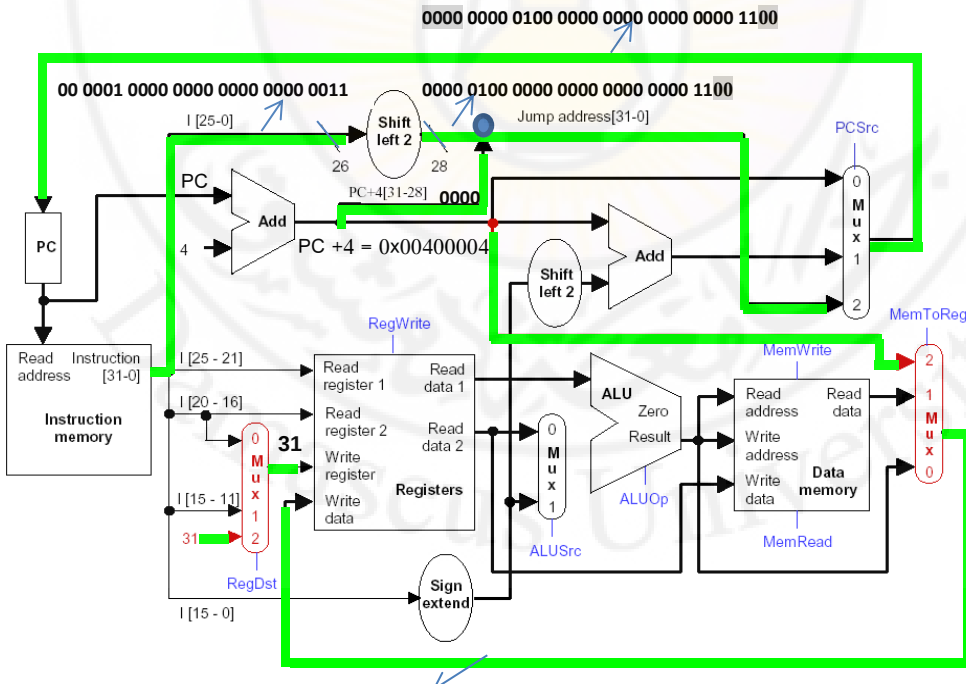
$0x00400000 = 100000000000000000000000_2$

عنوان التعليمة التالية لها:

$0x00400004 = 100000000000000000000000100_2$

عنوان تعليمة وجهة القفز:

$0x0040000C = 1000000000000000000000001100_2$



$\text{PC} + 4 = 0x00400004 = 0000 \ 0000 \ 0100 \ 0000 \ 0000 \ 0000 \ 0000$

تقوم التعليم بالقفز إلى عنوان الوجهة Label، مع حفظ عنوان العودة PC+4 في سجل العودة ra=\$31.

المثال (5-7): وضّح مسار المعطيات متعدد الأدوار عند تنفيذ التعلّية التالية في كل دور من أدوارها:

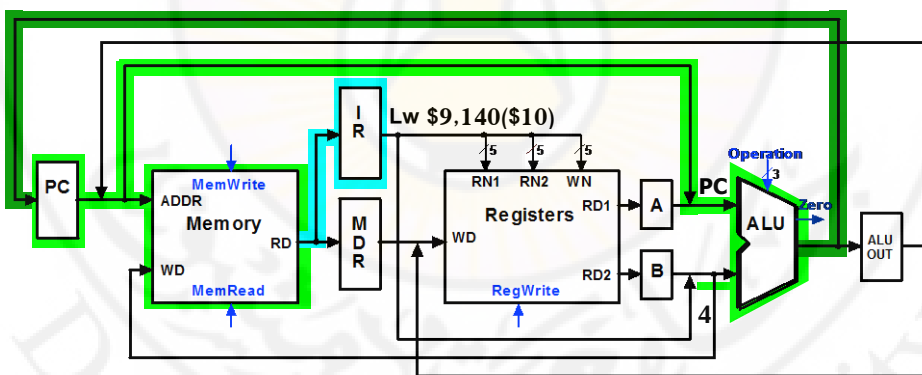
Lw \$9,140(\$10)

بفرض أن المحتوى الابتدائي للسجل $100 = \$10$ ومحتوى الذاكرة $M[240] = 50$.

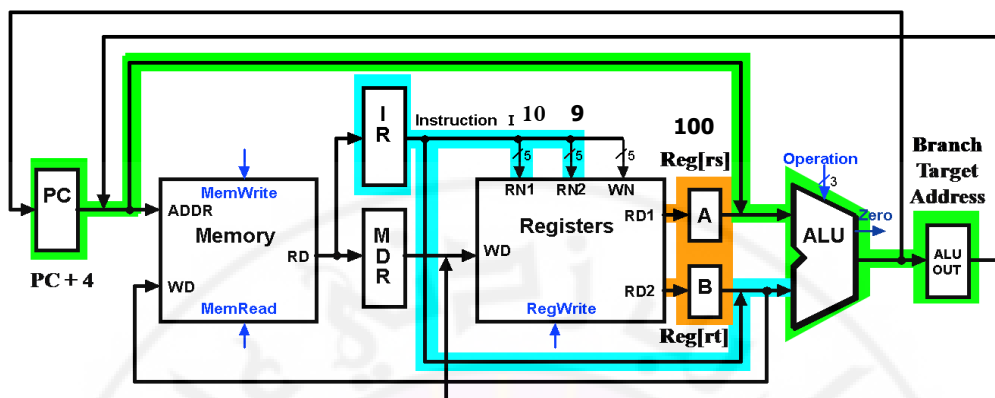
الحل:

المرحلة الأولى: جلب التعليمات fetch من الذاكرة إلى السجل IR وزيادة عدد البرنامج PC بمقدار 4.

```
IR = Memory[PC];  PC = PC + 4;
```

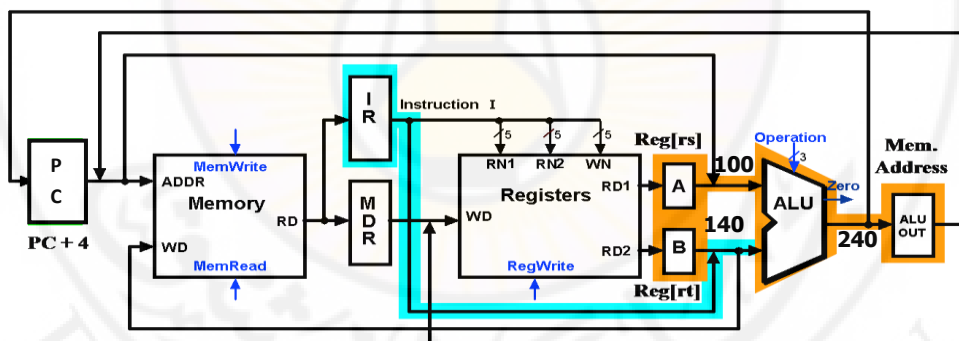


المرحلة الثانية: تحليل التعليمة decode وتحديد ماهيتها وما هي معاملاتها وقراءة محتوى السجل \$10 وتخزينه في السجل A. (وفي هذه المرحلة يتم أيضاً حساب عنوان التفرع باستخدام ALU، للاستفادة منه فيما لو كانت التعليمة هي تعليمة تفرع).



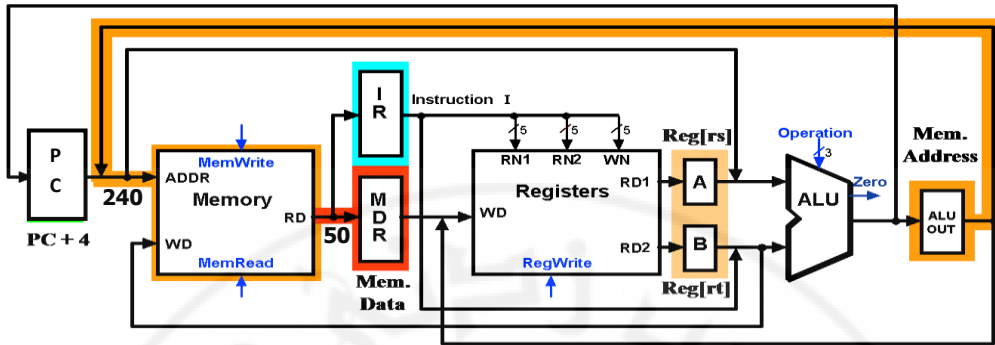
المرحلة الثالثة: حساب عنوان الذاكرة باستخدام ALU ووضع الناتج في السجل
.ALUOut

$$ALUOut = Mem @ = A + \text{sign-ext}(\text{imm}) = 100 + 140 = 240$$

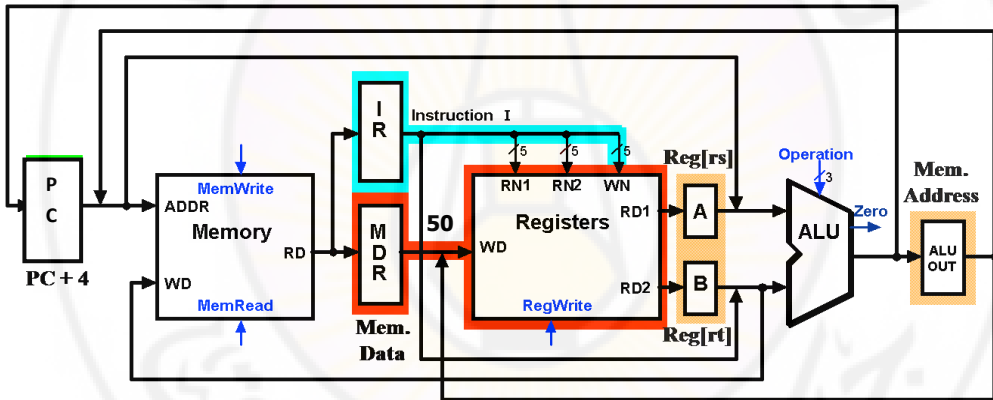


المرحلة الرابعة: يتم النفاذ إلى الذاكرة وقراءة القيمة المخزنة فيها عند العنوان
240 وحفظها في السجل MDR. أي:

$$MDR \leftarrow (MEM[240] = 50)$$



المرحلة الخامسة والأخيرة: نقل محتوى السجل MDR (القيمة 50) إلى السجل الوجهة في ملف السجلات \$9.



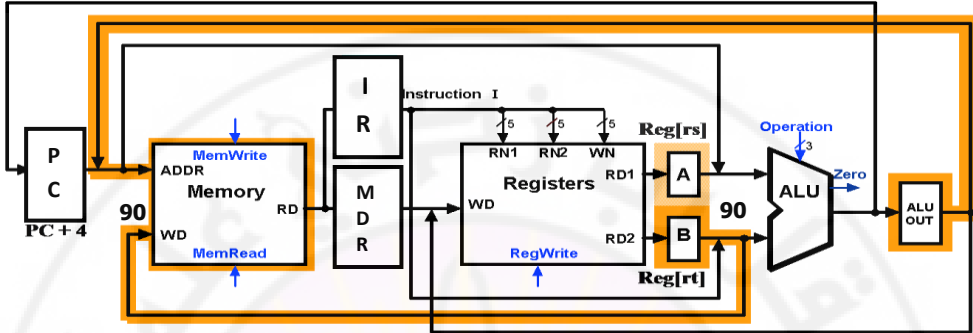
المثال (5-8): وضّح مسار المعطيات على المعالج متعدد الأدوار عند تنفيذ التعليمة (Sw \$9,140(\$10) بفرض أن المحتوى الابتدائي للسجلات \$10=100, \$9=90.

الحل:

المراحل الثلاثة للتعليمة Sw هي نفسها في تعليمة Lw من المثال السابق، أما المرحلة الرابعة وهي المرحلة الأخيرة فيتم فيها النفاذ إلى الذاكرة لتخزين محتوى السجل \$9 (المحفوظ في السجل المرحلي B بالمرحلة الثانية) في الذاكرة عند الموقع ذي العنوان M[240] الذي تم حسابه في المرحلة الثالثة لتنفيذ التعليمة:

$M[ALUOut] \leftarrow B$

$M[240] \leftarrow (\$9=90)$



المثال (5-9): لاحظنا في بنية معالج MIPS أن العمليات الحسابية تُجرى على سجلات وتحفظ النتائج في سجلات، ولكن لنفترض وجود التعليمة التالية:

$addm\ rd,\ rs,\ rt \quad \# \quad rd = rs + Mem[rt]$

أي أن السجل rt يحوي عنوان موقع الذاكرة الذي ستنتم قراءته وجمع محتواه مع السجل rs وحفظ النتيجة في السجل rd .

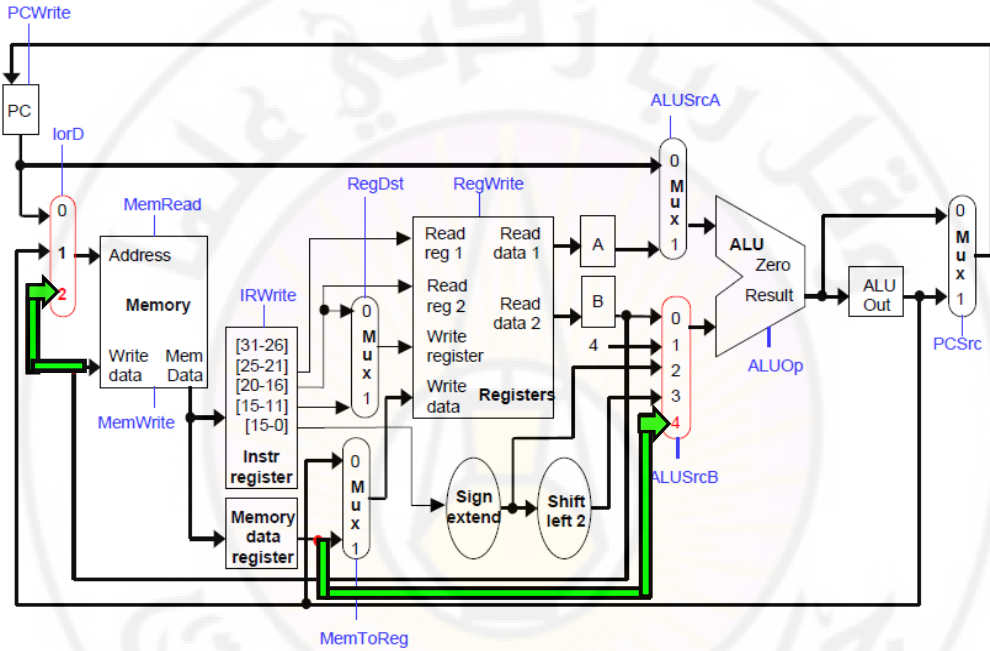
وبفرض أن صيغة التعليمة من النمط R-Type:

Field	op	rs	rt	rd	shamt	func
Bits	31-26	25-21	20-16	15-11	10-6	5-0

المطلوب: وضّح على ممر المعطيات متعدد الأدوار المرفق التعديلات المطلوبة لإمكانية تنفيذ هذه التعليمة، بشرط أن لا تحتاج التعليمة إلى أكثر من 5 مراحل للتنفيذ، وبشرط عدم إجراء تعديل على أي سجل ما عدا السجل rd ، يمكن إضافة خطوط أو نواخب أو تعديل عدد مداخل النواخب عند الحاجة.

الحل:

يلزم تأمين خط وصل بين السجل rt (الذي يحفظ في السجل B) ومدخل عنوانة الذاكرة Address، وكذلك خط وصل بين السجل MDR الذي يحفظ المعطيات المقروءة من الذاكرة، وبين المدخل الثاني لوحدة الحساب والمنطق ALU، كما هو موضح في المخطط التالي:



المثال (5-10): أ- وضّح ما هي المراحل التي تمر بها التعليمات التالية:
 Add, Addi, Sw, Lw, Beq, J من أجل معالج MIPS متعدد الأدوار، وما عدد أدوار التنفيذ في كل نوع منها؟
 ب- بفرض أن المعالج المذكور ينفذ برنامجاً مؤلفاً من تشكيلة تعليمات بالنسب التالية: تعليمات حسابية ومنطقية ALU (52%)، تعليمات تخزين في الذاكرة Sw (10%)، تعليمات تحميل من الذاكرة Lw (25%)، تعليمات تفرع (13%). احسب CPI الوسطي لهذا البرنامج.
الحل:

أ- المراحل التي تمر بها التعليمات:

مراحل التعليمات	Add, (R-Type)	Lw, Sw	Beq	J
جلب التعليمات	IR = Memory[PC]			
زيادة PC	PC = PC + 4			
تحليل التعليمات	A = Reg [IR[25-21]]			
وقراءة السجلات	B = Reg [IR[20-16]]			
وحساب عنوان التفرع	ALUOut = PC + (sign-extend (IR[15-0]) << 2)			
تنفيذ / عنوان الذاكرة	ALUOut=A op B	ALUOut=	if (A ==B) then	PC = PC [31-28]
إتمام التفرع/ القفر		A + s.ext(IR[15-0])	PC = ALUOut	(IR[25-0]<<2)
إتمام تعليمات add	Reg [IR[15-11]]	Lw: MDR = M[ALUOut]		
أو نفاذ إلى الذاكرة	ALUOut	Sw: M [AUOut] = B		
إتمام القراءة من الذاكرة		Lw: Reg[IR[20-16]]=MDR		
عدد أدوار التعليمات	4	4 (Sw) , 5 (Lw)	3	3

ب- حساب عدد الأدوار الوسطي في التعليمات CPI_{avg} :

$$CPI_{avg} = 0.52 \times 4 + 0.1 \times 4 + 0.25 \times 5 + 0.13 \times 3 = 4.12$$

المثال (5-11): بفرض أن لدينا معالج أحادي الدور فيه أزمدة التأخير كما يلي:

القراءة من ذاكرة التعليمات 2ns، القراءة من ملف السجلات 1ns، زمن ALU عند الحساب 2ns، زمن النفاذ لذاكرة المعطيات للقراءة أو الكتابة 2ns، زمن الكتابة في ملف السجلات 1ns.

المطلوب:

أ- ما هو الزمن الفعلي الذي يحتاجه تنفيذ كل من: add, sw, lw, beq؟

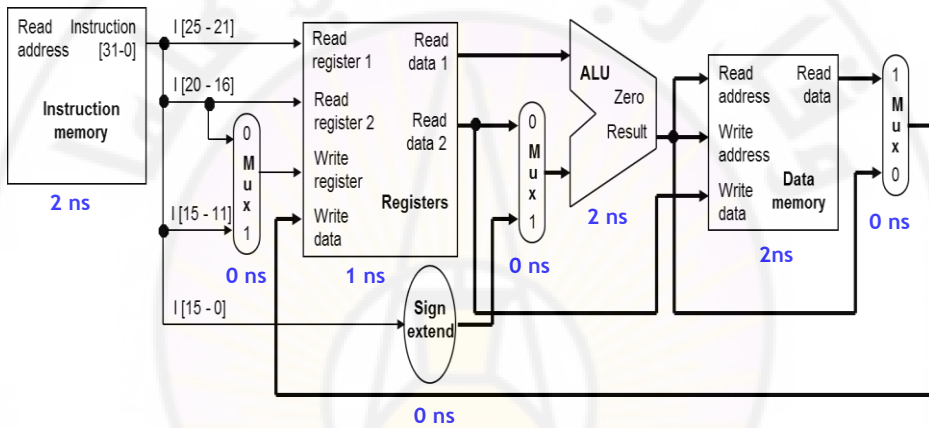
ب- ما هو دور الساعة (الأصغري) لهذا المعالج أحادي الدور بفرض أن

التعليمات التي ينفذها هي التعليمات السابقة؟

ج- ما هو دور الساعة (الأصغري) في حال كون المعالج متعدد الأدوار من أجل نفس أزمدة التأخير المذكورة في المسألة؟

د- احسب زمن تنفيذ كل من التعليمات lw، beq، add في حالة ممر المعطيات متعدد الأدوار؟

الحل: أ-



$$\text{Add: } 2+1+2+1= 6\text{ns}$$

$$\text{Sw: } 2+1+2+2= 7\text{ns}$$

$$\text{Lw: } 2+1+2+2+1= 8\text{ns}$$

$$\text{Beq: } 2+1+2= 5\text{ns}$$

ب- في المعالج أحادي الدور فإن كل تعليمة يجب أن يكتمل تنفيذها خلال دور ساعة واحد، لذا فإن زمن دور الساعة يجب أن يكون كافياً لإتمام أبطأ تعليمة، وهي في حالتنا تعليمة LW التي تحتاج على الأقل إلى 8ns لإتمام تنفيذها، وبالتالي فإن دور الساعة يجب أن لا يقل عن 8ns، أي:

$$\text{Cycle Time } (T)_{s.c} = 8\text{ns}$$

ولذلك فإن كل تعليمة ستأخذ 8ns في تنفيذها حتى وإن لم تكن تحتاج كامل هذا الوقت.

ج- في المعالج متعدد الأدوار فإن زمن دور الساعة يجب أن يكون كافياً لإتمام أبطأ مرحلة من مراحل التعليمات، وفي هذه الحالة فإن أبطأ مرحلة تستغرق 2ns، وبالتالي فإن:

$$\text{Cycle Time } (T)_{M.C} = 2ns$$

د- زمن تنفيذ التعليمات:

$$Lw: 2ns \times 5 = 10ns$$

$$Beq: 2ns \times 3 = 6ns$$

$$Add: 2ns \times 4 = 8ns$$

المثال (5-12): بفرض البرنامج التالي المكتوب بلغة التجميع MIPS لحلقة تقوم بضرب كل عنصر من الشعاع A بالقيمة s المخزنة في \$a3 ، طول الشعاع في \$t2:

```
loop: lw    $t0, 0($a0)
      mul   $t0, $t0, $a3
      sw    $t0, 0($a1)
      add   $a0, $a0, 4
      add   $a1, $a1, 4
      add   $t1, $t1, 1
      blt   $t1, $t2, loop
```

المطلوب: أ- ما هو عدد الأدوار الكلي لتكرار واحد من هذه الحلقة عند التنفيذ على مسار معطيات أحادي الدور؟

ب- ما هو عدد الأدوار الكلي لتكرار واحد من هذه الحلقة عند التنفيذ على مسار معطيات متعدد الأدوار؟ افترض بأن زمن التعليمة mul يساوي زمن تعليمة add، وزمن التعليمة blt يساوي زمن التعليمة beq.

الحل:

أ- كل تعليمة تُنفذ في دور ساعة واحد، ولدينا 7 تعليمات في الحلقة وبالتالي يلزمها 7 أدوار من أجل تكرار واحد للحلقة.

ب- نحدد عدد الأدوار في كل نوع من التعليمات المستخدمة: Lw (5cycle) ، blt (3cycle) ، add (4cycle) ، sw (4cycle) ، mul (4cycle)

وبالتالي فإن عدد الأدوار الكلي لتكرار واحد من هذه الحلقة على مسار معطيات متعدد الأدوار:

$$5 + 4 + 4 + 4 + 4 + 4 + 3 = 28 \text{ cycles}$$

المثال (5-13): بفرض لدينا معالج يحوي وحدة حساب إضافية لأعداد الفاصلة العائمة، وبفرض أن التأخيرات الزمنية للوحدات الوظيفية كما يلي:

Memory=2ns, ALU=2ns, FPUadd=8ns, FPUmultiply=16ns, register file read or write= 1 ns,

ومع إهمال التأخير الزمني للنواخب ووحدة التحكم والتمديد وخطوط التوصيل.

فإذا كانت نسب التعليمات في البرنامج المنفذ هي:

Load: 31% , Store: 21%, R-type Inst. : 27%, Branches: 5%, jump: 2%, FP add/ subtract: 7%, FP multiply and divide: 7%.

وبفرض إمكانية جعل دور الساعة مختلفاً $T_{variable}$ بحيث كل تعليمة تنفذ خلال دور واحد مقداره الزمن الذي تحتاجه التعليمة ليكتمل تنفيذها، قارن الأداء بين هذه الحالة (الافتراضية) وبين حالة ممر معطيات أحادي الدور بزمان دور ساعة ثابت T_{fixed} .

الحل:

التعليمة	Inst Mem	Reg. read	ALU	Data Mem	Reg. write	FPU add/sub	FPU mul/div	Total time
Load	2	1	2	2	1	0	0	8
Store	2	1	2	2	0	0	0	7
R-type	2	1	2	0	1	0	0	6
Branch	2	1	2	0	0	0	0	5
Jump	2	0	0	0	0	0	0	2
FPadd	2	1	0	0	1	8	0	12
FPmul	2	1	0	0	1	0	16	20

زمن دور الساعة في حالة الدور الثابت هو زمن أطول تعليمة، وفي هذه الحالة فإن أطول تعليمة هي تعليمة الضرب والقسمة لأعداد الفاصلة العائمة، إذاً:

$$T_{fixed-period} = 20ns$$

زمن دور الساعة الوسطي في حالة الدور المتغير:

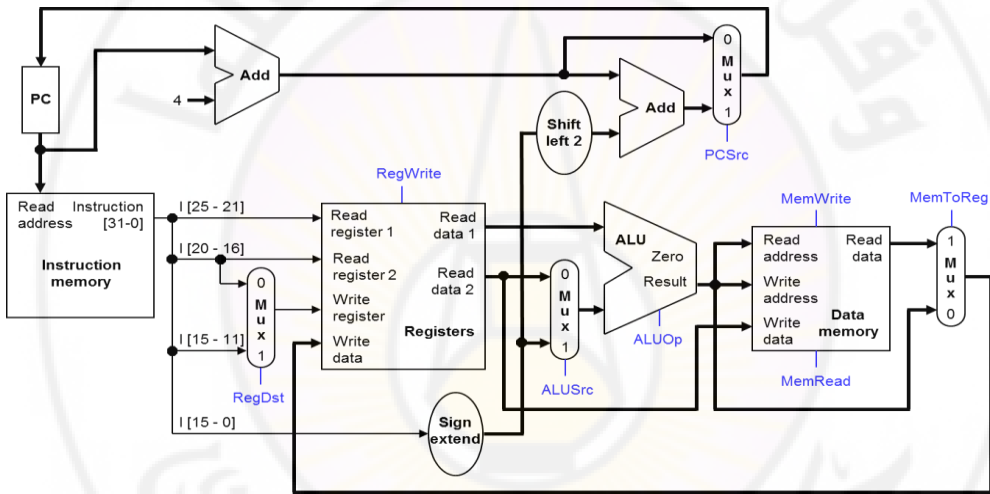
$$T_{var} = 8 \times 31\% + 7 \times 21\% + 6 \times 27\% + 5 \times 5\% + 2 \times 2\% + 12 \times 7\% + 20 \times 7\%$$

$$T_{var} = 2.4 + 1.47 + 1.62 + 0.25 + 0.04 + 0.84 + 1.4 = 8.1 \text{ ns}$$

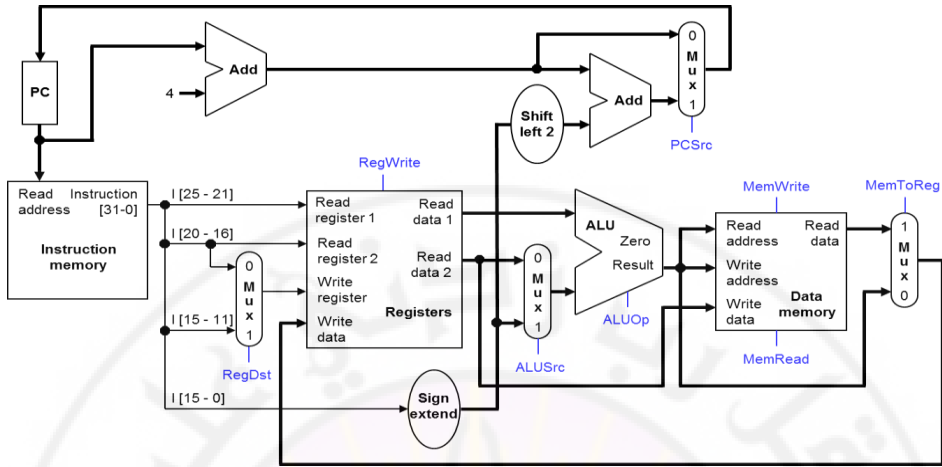
$$\text{performance}_{var}/\text{performance}_{fixed} = \text{CPUt}_{fixed}/\text{CPUt}_{var} = 20/8.1 = 2.5$$

تمارين غير محلولة

التمرين (5-1): وضّح ممر المعطيات على معالج MIPS أحادي الدور عند تنفيذ التعليمة \$12,\$13,\$14 and بفرض أن القيم الابتدائية للسجلات المستخدمة: \$12=5, \$13=2, \$14=7، ما هي القيم عند مداخل ومخارج ملف السجلات ووحدة الحساب والمنطق، وما القيمة النهائية التي يتم كتابتها؟ وأين تُكتب؟

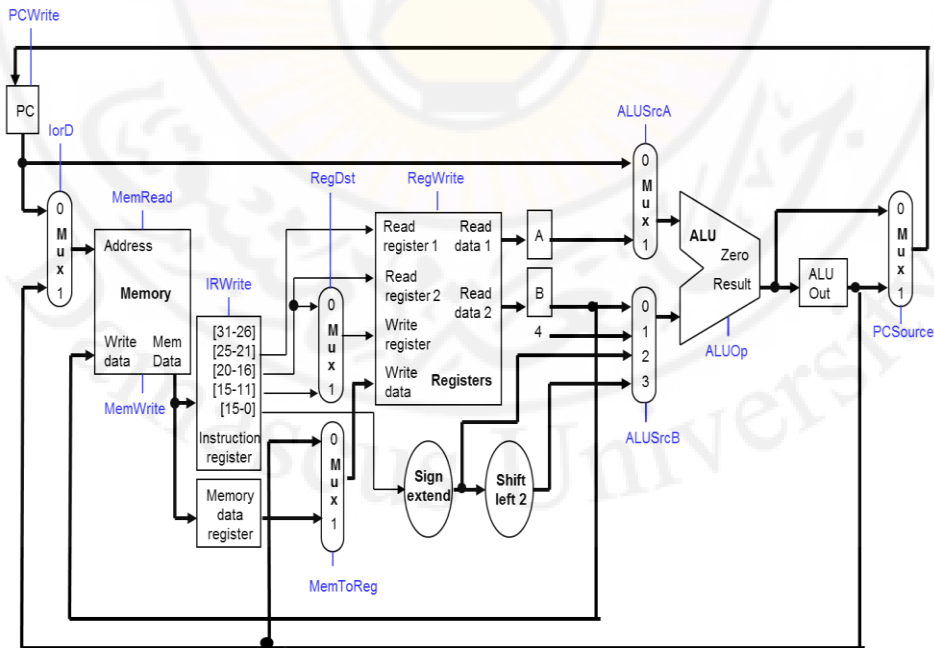


التمرين (5-2): وضّح على ممر المعطيات أحادي الدور التالي ما هي التعديلات المطلوبة لإمكانية تنفيذ تعليمة القفز إلى التعليمة ذات العنوان المخزن في السجل rs: (jr rs) علماً أنها من النمط R-Type. يمكن إضافة خطوط أو نواخب أو تعديل عدد مداخل النواخب عند الحاجة.

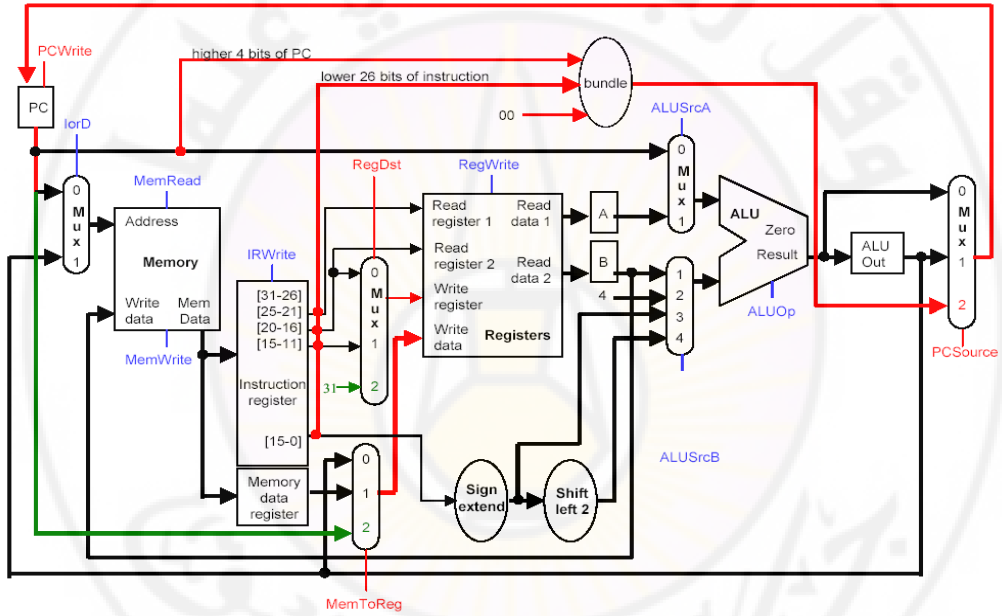


التمرين (3-5): وضّح القيم العددية على مسار المعطيات متعدد الأدوار التالي عند تنفيذ التعليمة: Sub \$9,\$10,\$11، بفرض أن محتوى السجلات الابتدائي كما يلي: \$9=50, \$10=60, \$11=40.

أين يتم حفظ القيم في كل دور من أدوار التعليمة؟



التمرين (5-4): وضّح مسار المعطيات من أجل معالج MIPS متعدد الأدوار عند تنفيذ التعليمة: jal Label إذا علمت أن عنوان هذه التعليمة هو 0x00400008 وأن عنوان وجهة القفز Label هو 0x00400020. وحدّد على المخطط المرفق كيفية الحصول على عنوان القفز كاملاً. وما هي القيمة التي يتم كتابتها على ملف السجلات؟



التمرين (5-5): بفرض لدينا برنامج مؤلف من النسب التالية للتعليّيات: 22%loads, 11%stores, 49%R-type, 16%branch, 2%jumps يتم تنفيذه على معالج MIPS بممر معطيات متعدد الأدوار MC، المطلوب: احسب CPI الوسطي (بفرض أن كل مرحلة تحتاج دور ساعة واحد).

التمرين (5-6): بفرض أن أزمّة التأخير في ممر المعطيات كانت كما يلي:

زمن النفاذ إلى الذاكرة = 190 ps، زمن النفاذ إلى ملف السجلات (قراءة/ كتابة) = 150 ps، زمن الحساب في ALU للتعليمات الأساسية = 190 ps، زمن الحساب في ALU لتعليمات الضرب والقسمة = 550 ps، وبإهمال بقية أزمنة التأخير.

وبفرض أنه يتم تنفيذ برنامج فيه مزيج التعليمات بالنسب التالية:

30% ALU, 15% mult & div, 15% load, 15% store, 15% branch, 10% jump.

المطلوب:

- أ- ما هو الزمن الفعلي اللازم لتنفيذ كل تعليمة من التعليمات السابقة؟
- ب- ما هو دور الساعة بفرض أن ممر المعطيات أحادي الدور.
- ج- بفرض أن ممر المعطيات متعدد الأدوار، وفيه دور الساعة 200 ps، وبفرض إمكانية إجراء عمليات الضرب والقسمة خلال عدة أدوار، احسب CPI لكل نوع من التعليمات السابقة، ثم احسب التسريع الحاصل speedup في هذه الحالة مقارنة مع الحالة ب.



الفصل السادس

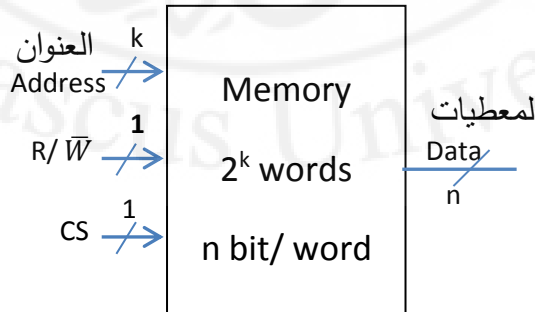
توسيع الذاكر Memory Expansion

مقدمة:

الذاكرة عبارة عن مجموعة خلايا cells للتخزين، كل خلية تخزن بتاً واحداً، وتتألف الذاكرة من مجموعة كلمات words، تمثل الكلمة إما تعليمة أو معطيات من نوع ما، وكل كلمة في الذاكرة لها عنوان فريد unique، ويتم الوصول إلى عنصر معين من الذاكرة عن طريق عنوان الذاكرة، ويسمى العدد الكلي لمواقع الذاكرة التي يمكن عنوانها بفضاء العنوان address space، بينما سعة الذاكرة Capacity هي عدد البتات الكلية التي يمكن تخزينها في هذه الذاكرة. ويتم التعبير عن أبعاد الذاكرة بالشكل $(m \times n)$ حيث m يمثل عدد الكلمات في الذاكرة أو طول الذاكرة، أما n فيمثل عرض الذاكرة أو عرض الكلمة في الذاكرة.

العمليات التي يمكن أن تتم على الذاكرة هي القراءة والكتابة لعنصر معطيات معين (مثلاً بت أو بايت أو كلمة.. إلخ).

وفيما يلي الشكل العام للذاكرة، وفيه: كلمة الذاكرة مؤلفة من n بت، عنوان الذاكرة مؤلف من k خط، يمكنه عنوان $2^k = \max m$ كلمة في الذاكرة، علماً أن عناوين الذاكرة هي أعداد ثنائية صحيحة موجبة unsigned.



تتم قراءة المعطيات من الذاكرة بوضع العنوان المطلوب على خطوط العنوان، وتفعيل خط التحكم بالقراءة Read وانتظار استقرار المعطيات المراد قراءتها. بينما تتم الكتابة عليها بوضع العنوان المطلوب على خطوط العنوان، ووضع المعطيات المطلوب كتابتها على خطوط المعطيات Data، وتفعيل خط التحكم بالكتابة Write، وغالباً يوجد خط انتقاء لرقاقة الذاكرة Chip Select أو يسمى خط تأهيل Enable، مع خط تحكم واحد يحدد إما عملية القراءة أو الكتابة على الذاكرة Read/ Write. ويبين الجدول التالي دور خطوط التحكم في الذاكرة:

العملية التي تتم على الذاكرة	R/ $\bar{W}$	CS أو E
لا يتم أي عملية/ الذاكرة غير مؤهلة	x	0
كتابة على الذاكرة	0	1
قراءة من الذاكرة	1	1

ويتناول هذا الفصل كيفية توسيع الذاكر، أي الحصول على ذاكرة أكبر من عدة ذواكر صغيرة، إما بزيادة عرض كلمة الذاكرة ويسمى توسيع عرضي Expanding memory width، أو بزيادة عدد أسطر الذاكرة ويسمى توسيع طولي Expanding memory length، أو باستخدام التوسيع طولاً وعرضاً في نفس الوقت، ويتم ذلك بوصل معين بين خطوط العنوان وخطوط المعطيات وخطوط التحكم حسب التوسيع المطلوب.

والجدول التالي يوضح عرض خط العنوان المؤلف من k بت وما يقابله من عدد كلمات الذاكرة التي يمكن عنوانتها به، وتحسب بالعلاقة $\max m = 2^k$.

max m	k (بت)	max m	k (بت)	max m	k (بت)
2M	21	2k	11	2	1
4M	22	4k	12	4	2
8M	23	8k	13	8	3
16M	24	16k	14	16	4
32M	25	32k	15	32	5
64M	26	64k	16	64	6
128M	27	128k	17	128	7
256M	28	256k	18	256	8
512M	29	512k	19	512	9
1024M=1G	30	1024k=1M	20	1024=1K	10

أمثلة محلولة:

المثال (6-1): احسب عدد خطوط العنونة من أجل ذاكرة بحجم 64K Byte
إذا علمت أن الكلمة مؤلفة من 1 Byte، وما هو فضاء العناوين لهذه الذاكرة؟
الحل:

بما أن الكلمة مؤلفة من 1 Byte، فإن عدد كلمات الذاكرة هو 64K Word
فيلزمها خط عنونة 16، حيث أن $2^{16} = 64K$.
فضاء العناوين من العنوان 0 وحتى العنوان $2^{16} - 1 = 65535$.

المثال (6-2): ذاكرة مؤلفة من 1Kword، وعرض الكلمة هو 20 بت، ما هو
حجم الذاكرة مقدراً بالبايت؟
الحل:

$$\begin{aligned} 1 \text{ Byte} &= 8 \text{ bit} = 2^3 \text{ bit} \\ 20 * 1K &= 20 \text{ Kbit} = 20 * 2^{10} \text{ bit} = 20 * 2^7 \text{ Byte} = 20 * 128 \text{ Byte} \\ &= 2560 \text{ Byte} \end{aligned}$$

المثال (6-3): بفرض أن حجم الذاكرة في نظام حاسوبي هو 32MB، ما عدد
البتات اللازمة لعنونة أي بايت في الذاكرة؟
الحل:

$$32 \text{ MB} = 2^5 \times 2^{20} = 2^{25} \text{ Byte}$$

وبالتالي يلزم $\log_2 2^{25} = 25 \text{ bits}$ لعنونة أي بايت ضمن الذاكرة.

المثال (6-4): بفرض أن حجم الذاكرة في نظام حاسوبي هو 128MB، وكل
كلمة مؤلفة من 8 Byte، ما عدد البتات اللازمة لعنونة أي كلمة في الذاكرة؟
الحل:

$$1 \text{ word} = 8 \text{ Byte} = 2^3 \text{ Byte}$$

$$128 \text{ MB} = 2^7 \times 2^{20} \text{ Byte} = 2^{27} \text{ Byte} = 2^{24} \text{ word}$$

وبالتالي يلزم $\log_2 2^{24} = 24 \text{ bits}$ لعنونة أي كلمة ضمن الذاكرة.

المثال (6-5): بفرض لدينا ذاكرة (2Kx8)، ما هو عدد الكلمات التي يمكن تخزينها في هذه الذاكرة، وكم يلزمها خط عنونة، وما هو الحجم الكلي للذاكرة بالبت؟
الحل:

عدد الكلمات التي يمكن تخزينها في هذه الذاكرة هو 2k أي 2x1024 وتساوي 2048 كلمة. يلزم لعنونها 11 خط عنونة، علماً أن كل كلمة عبارة عن 8 بت أي بايت واحد.

$$\text{الحجم الكلي للذاكرة بالبت: } 2048 \times 8 = 16384 \text{ bit}$$

المثال (6-6): أي من الذاكرتين التاليتين تخزن عدد بتات أكثر: M1(5Mx8) أم M2(1Mx16)؟
الحل:

$$M1: 5M \times 8 = 5 \times 1048576 \times 8 = 41\,943\,040 \text{ bit}$$

$$M2: 1M \times 16 = 1048576 \times 16 = 16\,777\,216 \text{ bit}$$

وبالتالي فإن الذاكرة M1 هي التي تخزن عدد بتات أكثر.

المثال (6-7): يراد تجميع عدة ذواكر سعتها 2Kx8 للحصول على ذاكرة بسعة 8Kx8. كم رقاقة ذاكرة يلزم في هذه الحالة؟ وما عدد خطوط العنونة المطلوبة؟ وما الحجم الكلي للذاكرة الناتجة بالبت؟
الحل:

يلزمنا 4 رقاقات ذاكرة بسعة 2Kx8، وباستخدام التوسيع طويلاً نحصل على

ذاكرة بسعة 8Kx8.

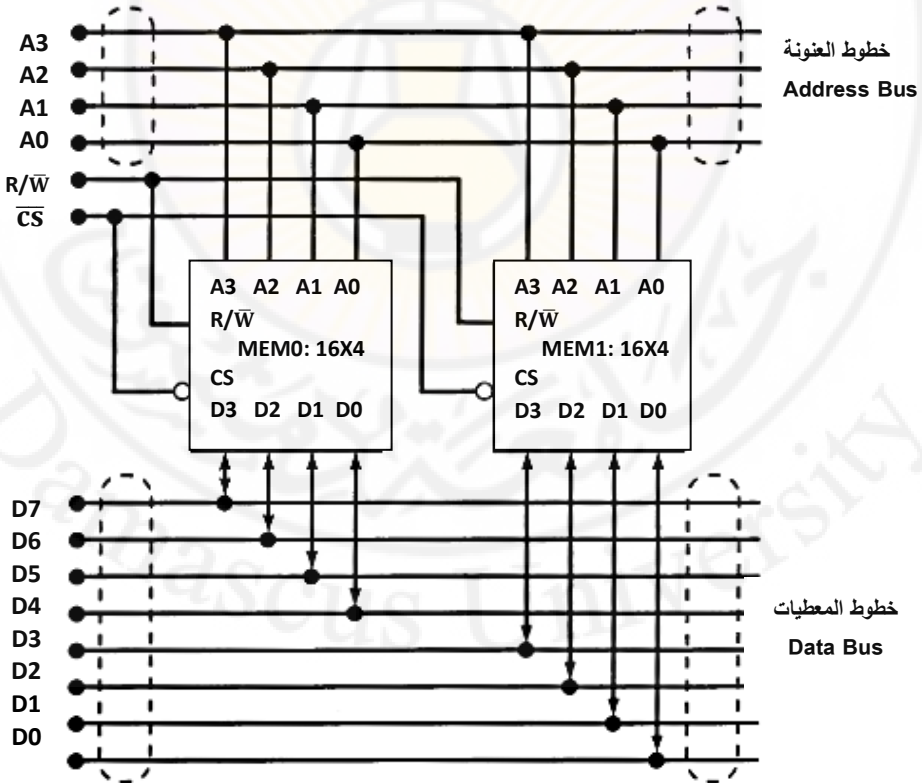
عدد خطوط العنوان للذاكرة الناتجة هو $\log_2 (8k)$ ويساوي 13 خط عنوان.

حجم الذاكرة الناتجة: $8 \times 1024 = 8192 \text{ bit}$.

المثال (6-8): نود بناء ذاكرة بسعة 16 كلمة، وعرض الكلمة 8 بت، انطلاقاً من ذواكر متوفرة بسعة 16×4 . ما عدد الذواكر اللازمة، وكيف يتم التوصيل بينها للحصول على المطلوب، علماً أن خطوط المعطيات مشتركة للقراءة والكتابة.

الحل:

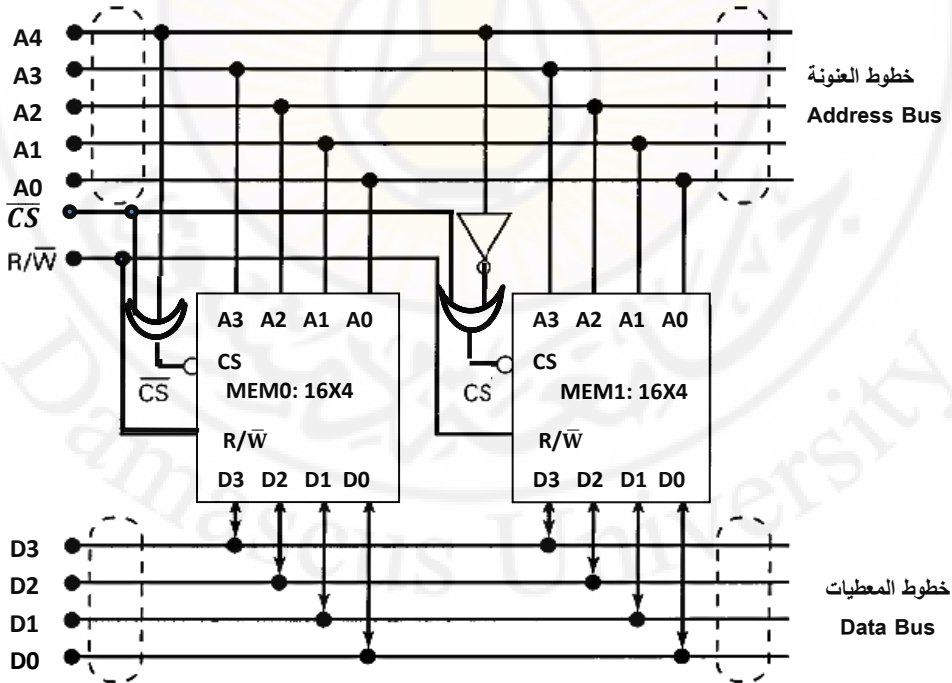
سعة الذواكر المتوفرة 16×4 ، وسعة الذاكرة المطلوبة 16×8 ، يتم الحصول عليها بالتوسيع العرضي باستخدام ذاكرتين بسعة 16×4 ، كما في الشكل التالي:



عرض الكلمات في الذاكرة الناتجة هو 8 بت، ونلاحظ أننا استخدمنا كل ذاكرة لتخزين نصف كلمة، الذاكرة MEM0 تخزن البتات الأربعة ذات الأوزان العليا في الكلمات، بينما الذاكرة MEM1 تخزن البتات الأربعة ذات الأوزان الدنيا في الكلمات. عدد الكلمات في الذاكرة الناتجة هو 16 كلمة، يتم عنوانها بأربعة خطوط عنوانية A0, A1, A2, A3. وقد تم ربط خطوط العنوانية والتحكم للذاكرتين معاً، بينما بقيت خطوط المعطيات منفصلة وبمجموعها حصلنا على عرض الكلمة المطلوبة.

المثال (6-9): نود بناء ذاكرة بسعة 32 كلمة، وعرض الكلمة 4 بت، انطلاقاً من ذواكر متوفرة بسعة 4 x 16، ما عدد الذواكر اللازمة، وكيف يتم التوصيل بينها للحصول على المطلوب، علماً أن خطوط المعطيات مشتركة للقراءة والكتابة.

الحل:



نحصل على الذاكرة المطلوبة بالتوسيع الطولي (زيادة عدد مواقع الذاكرة) باستخدام ذاكرتين سعة كل منهما 16x4.

مجال العناوين في الذاكرة الناتجة لعنونة 32 كلمة هو من 00000 إلى 11111، ونلاحظ أن:

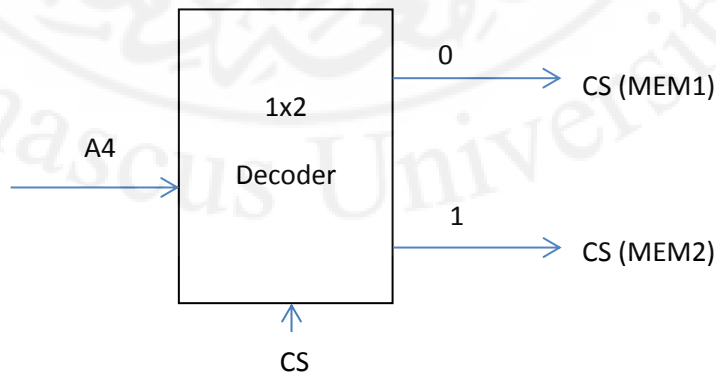
الذاكرة MEM0 تفعل من أجل مجال العناوين من 00000 إلى 01111.

الذاكرة MEM1 تفعل من أجل مجال العناوين من 10000 إلى 11111.

حيث أن خط الانتقاء للذاكر CS يفعل عند 1، بينما يتم إرسال $\bar{CS}$ (يدل وجود الخط فوق CS على أن التفعيل عند 0). ولا يتم تفعيل إلا ذاكرة واحدة فقط من الذاكرتين في الوقت نفسه.

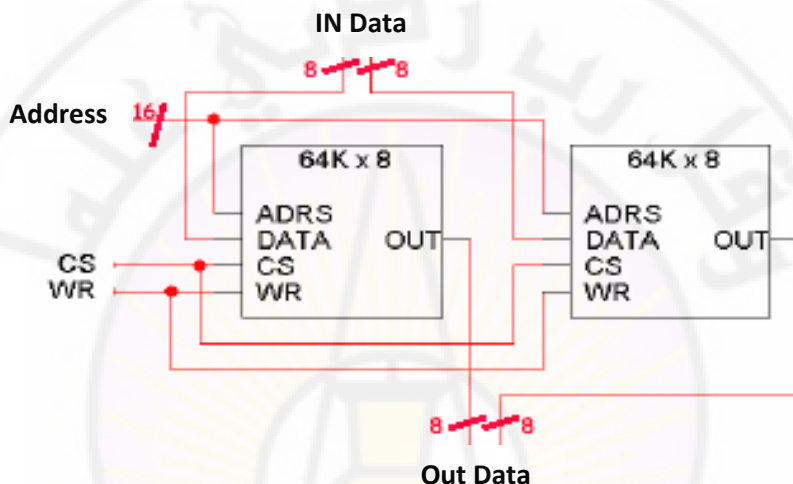
وقد تم ربط خطوط المعطيات للذاكرتين MEM0, MEM1 معاً، بينما تم توصيل خطوط العنونة الأربع A3, A2, A1, A0 إلى مداخل عنونة كل من الذاكرتين، أما خط العنونة A4 فيتم وصله إلى مدخل الانتقاء CS للذاكرتين، لإحدى الذاكرتين مع استخدام عاكس، وللأخرى بدون استخدامه، وبذلك يمكن الوصول إلى كامل العناوين في فضاء العنونة المتاح.

ملاحظة: يمكن استخدام مفكك ترميز Decoder مدخله هو خط العنونة A4 وله مخرجان يوصل كل منهما إلى إحدى الذاكرتين المستخدمتين:



المثال (6-10): نود بناء ذاكرة بسعة 64K x 16 RAM انطلاقاً من ذواكر بسعة 64K x 8، علماً أن مداخل المعطيات منفصلة عن مخرجها، وضح طريقة التوصيل.

الحل:

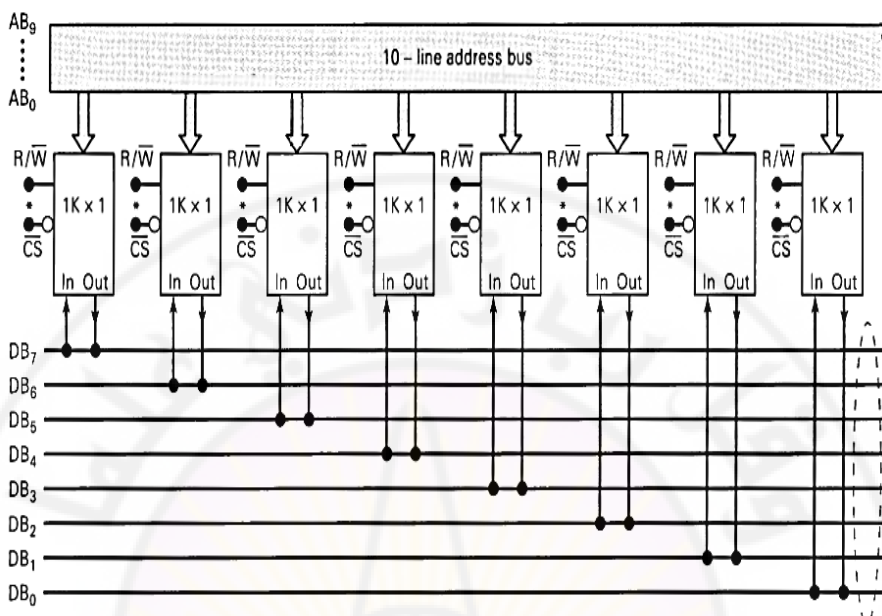


المثال (6-11): نود بناء ذاكرة بسعة 1Kx8 انطلاقاً من ذواكر بسعة 1Kx1، كم عدد الذواكر التي نحتاجها، وكيف يتم التوصيل بينها - علماً أن مداخل المعطيات منفصلة عن مخرجها-؟

الحل:

نحتاج إلى 8 ذواكر بسعة 1Kx1 للحصول على ذاكرة 1Kx8، باستخدام التوسيع العرضي، وذلك بربط خطوط العنونة و خطوط التحكم معاً لكافة الذواكر، وبتجميع خطوط المعطيات جنباً إلى جنب concatenation للحصول على العرض المطلوب وهو 8 بت.

والشكل التالي يوضح طريقة توصيل الذواكر للحصول على الذاكرة المطلوبة:



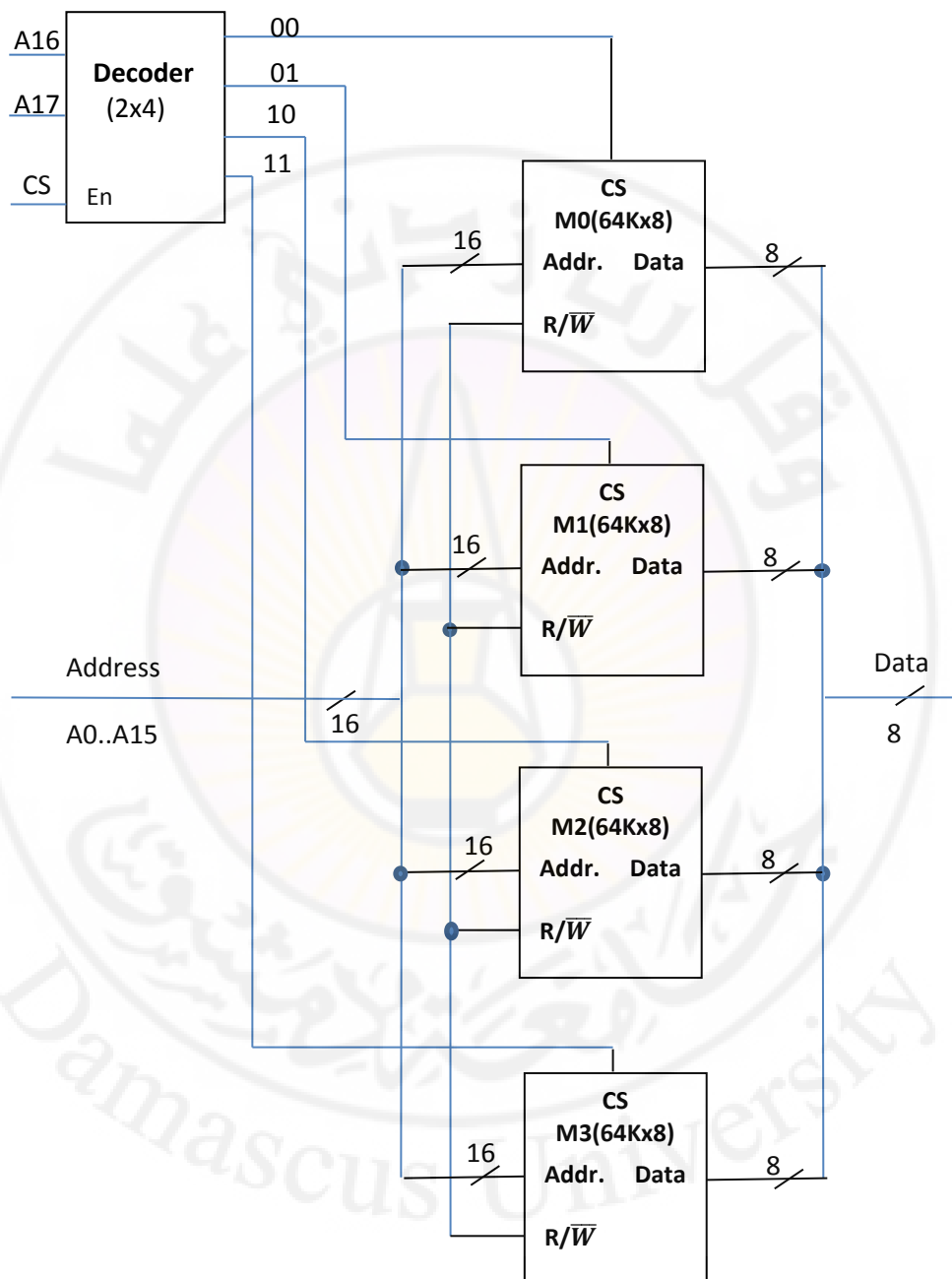
المثال (6-12): بفرض لدينا 4 ذواكر بسعة 64Kx8 يُراد الحصول منها على أطول ذاكرة ممكنة، وضح طريقة التوصيل، وما هو عدد خطوط العنوانو المعطيات وفضاء العناوين لكل من الذواكر المستخدمة وللذاكرة الناتجة.

الحل:

عدد خطوط العنوانو للذواكر المستخدمة هو 16 خط، وعرض كلماتها هو 8 بت.

نستخدم التوسيع طولاً للحصول على أكبر عدد مواقع في الذاكرة، انطلاقاً من الذواكر الأربعة ذات السعة 64Kx8، فنحصل على ذاكرة بسعة 256Kx8 عرض الكلمة فيها 8 بت، وعدد خطوط عنوانوها 18 خط.

يتم إدخال خطوط العنوانو ذات الأوزان العليا وهي في حالتنا A16, A17 إلى مفكك ترميز Decoder، وتوصل مخارجه إلى مداخل الانتقاء للذواكر الأربعة، بينما يتم وصل بقية خطوط العنوانو A0..A15 إلى مداخل العنوانو للذواكر الأربعة.



M0 00000000000000000000 وحتى 00 1111 1111 1111 1111 نُفَعْل

M1 01000000000000000000 وحتى 01 1111 1111 1111 1111 نُفَعْل

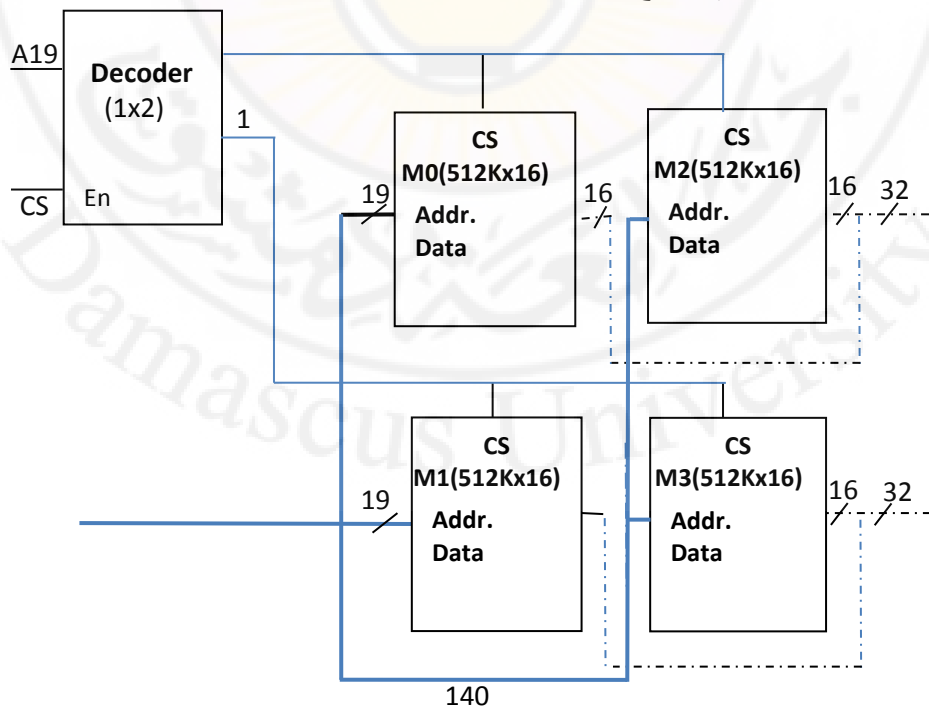
M2 10000000000000000000 وحتى 10 1111 1111 1111 1111 نُفَعْل

M3 11000000000000000000 وحتى 11 1111 1111 1111 1111 نُفَعْل

إذاً فضاء العناوين للذاكرة الناتجة M هو من:

00 0000 0000 0000 0000 وحتى 11 1111 1111 1111 1111 .

يلزمنا 4 ذواكر للحصول على الذاكرة المطلوبة، يتم تجميع M0,M2 عرضاً،
M1,M3 عرضاً، ثم تجميع الذاكرتين الناتجتين طولاً للحصول على الذاكرة المطلوبة:



إذا كان العنوان المطلوب من المجال 0000 0000 0000 0000 0000 إلى المجال 0111 1111 1111 1111 1111 فيتم تفعيل الذاكرتين M0,M2 معاً.

أما إذا كان العنوان المطلوب من المجال 1000 0000 0000 0000 0000 إلى المجال 1111 1111 1111 1111 1111 فيتم تفعيل الذاكرتين M1,M3 معاً.

المثال (6-14): ما هو عدد رقائق الذاكرة ROM ذات السعة 256x8 اللازمة للحصول على ذاكرة بسعة 4000 بايت؟ وكم عدد خطوط العنوان اللازمة للوصول إلى الذاكرة الناتجة؟ وما عدد خطوط العنوان المشتركة بين جميع هذه الرقائق؟
الحل:

حساب عدد رقائق الذاكرة اللازمة:

$$4000 \text{ Byte} \approx 4K \text{ Byte} = 2^2 \times 2^{10} = 2^{12} \text{ Byte}$$

$$\text{ROM size} = 256 \text{ Byte} = 2^8 \text{ Byte}$$

$$\text{No. of ROM chip} = 2^{12} / 2^8 = 2^4 = 16 \text{ chip}$$

عدد خطوط العنوان اللازمة للوصول إلى الذاكرة الناتجة (بحجم 4KB) = 12 خط.
عدد خطوط العنوان المشتركة بين جميع الرقائق (حسب حجم الرقاقة الواحدة) = 8 خط
عنوان.

المثال (6-15): بفرض لدينا الذاكر التالية:

M1(0.5k x16) , M2(0.5k x16) , M3(1k x16) , M4 (1k x16) ,
M5(1k x16) , M6(2k x 32).

ما هي طريقة التوصيل المتبعة للحصول على ذاكرة بسعة 4k*32؟

الحل:

للتبسيط نقوم بتجميع كل ذاكرتين معاً إلى أن نحصل على الذاكرة بالسعة المطلوبة $4k \times 32$:

بالتوسيع الطولي للذاكرتين $M1(0.5k \times 16)$, $M2(0.5k \times 16)$ نحصل على ذاكرة بسعة $M7(1k \times 16)$.

بالتوسيع العرضي للذاكرتين $M3(1k \times 16)$, $M4(1k \times 16)$ نحصل على ذاكرة بسعة $M8(1k \times 32)$.

بالتوسيع العرضي للذاكرتين $M7(1k \times 16)$, $M5(1k \times 16)$ نحصل على ذاكرة بسعة $M9(1k \times 32)$.

بالتوسيع الطولي للذاكرتين $M9(1k \times 32)$, $M8(1k \times 32)$ نحصل على ذاكرة بسعة $M10(2k \times 32)$.

بالتوسيع الطولي للذاكرتين $M6(2k \times 32)$, $M10(2k \times 32)$ نحصل على ذاكرة بسعة $M(4k \times 32)$ وهو المطلوب.

وقد تعرفنا على طريقة رسم المخطط الصندوقي في حالات التوسيع طولاً أو عرضاً من خلال الأمثلة المحلولة السابقة.

تمارين غير محلولة

التمرين (6-1): بفرض نظام ذاكرة فيه 32 خط معطيات، و 20 خط عنوان،

احسب كلاً مما يلي:

1- عدد مواقع الذاكرة التي يمكن عنوانتها.

2- عرض الذاكرة.

3- طول الذاكرة.

4- سعة الذاكرة بالبايت.

5- سعة الذاكرة بالبت.

التمرين (6-2): بفرض ذاكرة فيها 2 خط عنوان فقط، و 2 خط معطيات، ما

عدد البتات التي يمكن تخزينها في هذه الذاكرة؟

التمرين (6-3): كم عدد الذواكر ذات السعة (128x8) اللازمة للحصول على

ذاكرة بسعة 4096x16؟ وما هي طريقة التجميع المتبعة؟ ارسم المخطط الصندوقي للتوصيل.

التمرين (6-4): المطلوب بناء ذاكرة بسعة 32k word، بسعة كلمة 16 بت،

ويتوفر لدينا العديد من الذواكر بالسعات التالية: 16Kx8bits، 32Kx4bits،

8kx16bit، اختر طريقة تجميع للذواكر المتوفرة للحصول على الذاكرة المطلوبة، مع

رسم المخطط الصندوقي للتوصيل.

التمرين (6-5): بفرض أنه يتوفر لدينا 8 ذواكر سعة كل منها 16K x 32،

المطلوب توضيح طريقة تجميع هذه الذواكر ورسم المخطط الصندوقي للتوصيل

بينها وحساب سعة الذاكرة الناتجة في الحالتين التاليتين:

أ- للحصول على ذاكرة بأعرض كلمة معطيات ممكنة.

ب- للحصول على ذاكرة بأكثر عدد كلمات ممكن.

التمرين (6-6): بفرض نظام حاسوبي فيه خط العنوان مؤلف من 10 بت، وعرض خط المعطيات 1Byte، ويراد الحصول على كامل فضاء العناوين الممكن باستخدام رقاقات ذواكر بسعة 128x8.
المطلوب:

- أ- ما هو عدد الرقاقات التي نحتاجها للحصول على الذاكرة المطلوبة؟
ب- ارسم مخططاً صندوقياً لطريقة التوصيل مع توضيح عدد خطوط العنوان والمعطيات للذواكر، وما هي مداخل ومخارج مفكك الترميز المستخدم؟
ج- أعد الطالبين (أ و ب) بفرض أن عرض خط المعطيات هو 2Byte بدل 1Byte.

ملحق

دليل التعامل مع برنامج محاكاة معالج MIPS

مقدمة:

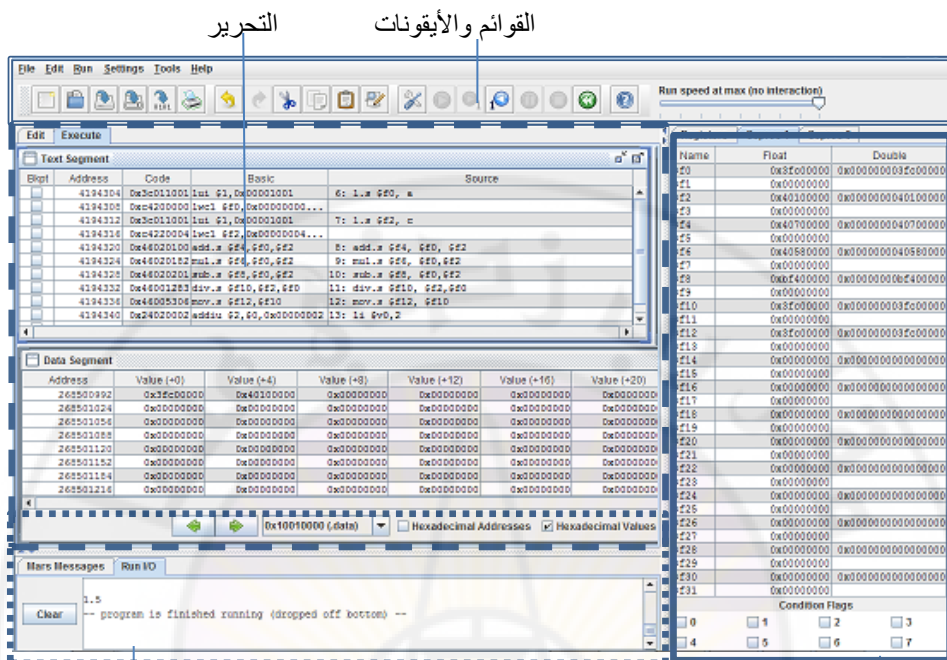
برنامج MARS هو برنامج محاكاة لتنفيذ البرامج المكتوبة بلغة التجميع Assembly من أجل معالجات MIPS، وهو اختصار للعبارة MIPS Assembler and Runtime Simulator، فهو إذاً مجّمع وبنفس الوقت برنامج محاكاة لمعالج MIPS وقت التنفيذ.

دخل برنامج المحاكاة هو برنامج بلغة التجميع وخرجه هو نتيجة تنفيذ البرنامج. ويمكن تحميل برنامج المحاكاة MARS من العنوان التالي:

www.cs.missouristate.edu/MARS

حيث تتوفر نسخ من البرنامج لكل من أنظمة التشغيل Linux، Windows، Mac.. إلخ، وهو مكتوب بلغة جافا، ويلزم توفر الإصدار 1.5 (أو ما بعده) من SDK J2SE Java على الجهاز حتى يعمل البرنامج، ويتم توزيعه على شكل ملف JAR قابل للتنفيذ.

بالنقر المزدوج على الأيقونة Mars_.jar تظهر واجهة البرنامج الرسومية كما في الشكل التالي:



رسائل البرنامج ودخل/ خرج

السجلات

القوائم والأيقونات: تستخدم للتحكم بمحرر النصوص، المجمع Assembler، وبرنامج محاكاة MIPS. ونلاحظ أن أغلب عناصر القوائم لها أيقونات مقابلة في شريط الأدوات، أو اختصارات للوحة المفاتيح.

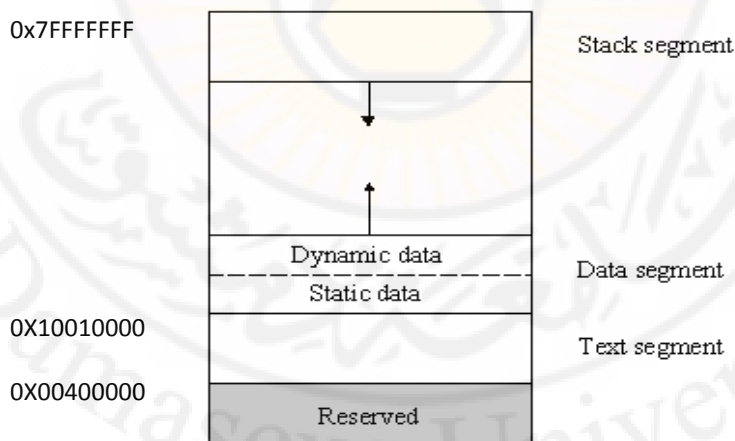
التحرير والتنفيذ: يستخدم التبويب Edit لكتابة وتعديل ملفات الشيفرة المصدرية المكتوبة بلغة التجميع، أما التبويب Execute فيستخدم لعرض الشيفرة بلغة الآلة وتنفيذ البرنامج وفحص محتوى ذاكرة المعطيات.

تبويب رسائل برنامج MARS، وتبويب دخل/ خرج التنفيذ: تعرض رسائل أخطاء المجمع، وأخطاء التنفيذ، ورسائل مختلفة عن حالة المحاكى، وعند الضغط على رسالة الخطأ يتم تحديد السطر الذي يحوي هذا الخطأ في محرر النصوص. ويعرض التبويب الثاني قيم الدخل والخرج أثناء تنفيذ البرنامج والتي تنتج عن تعليمات استدعاء النظام.

السجلات: وفيها ثلاثة تبويبات، تبويب RegisterS: وفيه سجلات الأعداد الصحيحة من \$0 وحتى \$31، وعداد البرنامج PC، والسجلان Hi,Lo المستخدمة في تعليمات الضرب والقسمة، والتبويب Coproc0 وفيه سجلات الاستثناءات والمقاطعات، والتبويب Coproc1 وفيه سجلات الأعداد ذات الفاصلة العائمة.

عند تنفيذ برنامج ما فإن بإمكانه الوصول إلى ثلاث مناطق من الذاكرة:

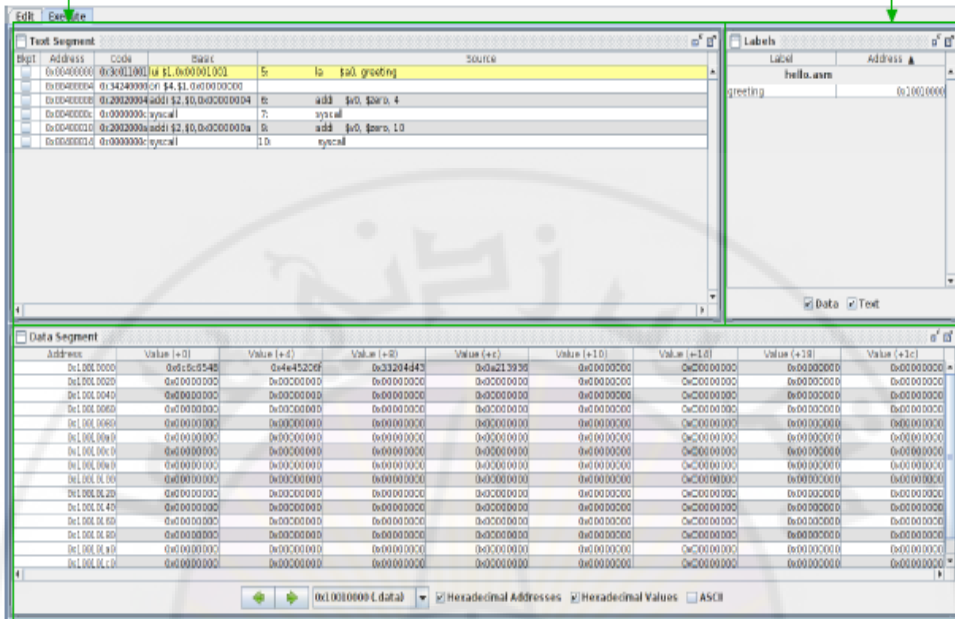
- 1- مقطع النص Text ويبدأ من العنوان 0x00400000 وبالتالي فإن عداد البرنامج PC يبدأ في جلب التعليمات وتنفيذ البرامج من هذا العنوان. ونستخدم هنا لفظ Text للتعبير عن البرنامج المكتوب بلغة الآلة (أصفار وواحدات) بدل استخدام لفظ Program المستعمل في لغات البرمجة عالية المستوى.
- 2- مقطع المعطيات Data: يحوي المتحولات العامة والثوابت المحرّفة والمعطيات، ويبدأ من العنوان 0x10010000.
- 3- المكسد Stack.



والشكل التالي يظهر واجهة التنفيذ Execute:

مقطع النص Text

التسميات Labels



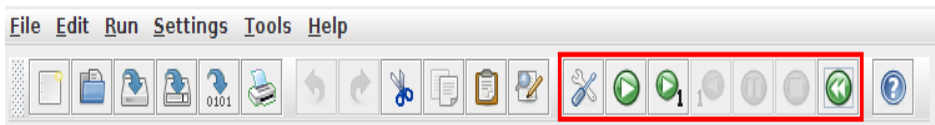
مقطع المعطيات Data Segment

مقطع النص Text: يعرض تعليمات البرامج بلغة الآلة وبلغة التجميع، ويسمح باستخدام نقاط التوقف breakpoints، ويميز التعليمة التالية التي سيتم جلبها وتنفيذها باللون الأصفر.

مقطع المعطيات Data: يظهر محتوى ذاكرة المعطيات، كل ثمان كلمات في سطر، ويمكن استخدام هذه النافذة لفحص مقطع data. والمكدس stack.

التسميات Labels: تعرض كل التسميات المستخدمة في البرنامج مع عناوينها المقابلة. وبالضغط على أية تسمية label أو العنوان المقابل لها يظهر محتوى هذا العنوان في مقطع المعطيات أو مقطع النص.

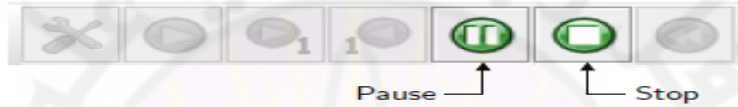
وفيما يلي بعض الأيقونات الهامة المستخدمة لتنفيذ البرامج:



1- الأزرار التي تكون مفعلة خارج فترة تنفيذ البرنامج :



2- الأزرار التي تكون مفعلة أثناء تنفيذ البرنامج :



الأيقونة Assemble: تستخدم عادة في تبويب التحرير Edit، تطلب ترجمة الشيفرة بلغة التجميع Assembly إلى لغة الآلة machine language، بحيث يمكن تنفيذ الشيفرة في التبويب Execute. وإذا كنت تستخدم محرر نصوص خارجياً لكتابة البرامج بلغة التجميع فيمكن اختيار تجميع الملف تلقائياً حالما يتم فتحه، وذلك من خلال قائمة الإعدادات Settings.

الأيقونة Run: تستخدم لتنفيذ البرنامج الذي تم تجميعه، أو لمتابعة تنفيذ برنامج تم إيقافه بنقطة توقف breakpoint أو التوقف المؤقت Pause.

الأيقونة Single-Step: تنفذ التعليمة التالية ثم يتوقف التنفيذ مؤقتاً.

الأيقونة Undo: للتراجع عن آخر تعليمة تم تنفيذها وإبطال التغييرات الحاصلة في السجلات والذاكرة، وهذه الميزة مفيدة جداً في برنامج المحاكاة، ولا تتوفر عموماً في المعالجات الحقيقية.

الأيقونة Reset: تعيد السجلات والذاكرة إلى قيمها الابتدائية.

الأيقونة Help: تفتح نافذة للمساعدة في استخدام برنامج المحاكاة MARS ومساعدة في لغة التجميع MIPS.

الأيقونة Pause: توقف البرنامج الذي يتم تنفيذه، ونقيد في حالة إمكانية دخول

البرنامج في حلقة غير منتهية.

الأيقونة Stop: تنهي البرنامج الذي يتم تنفيذه.

استدعاءات النظام **system calls**: ينقسم نظام التشغيل operating system بشكل عام إلى جزأين: فضاء النواة kernelSpace وفضاء المستخدم userSpace، أما فضاء النواة فهو الجزء الذي يتعامل مع العتاد hardware مباشرة، بينما فضاء المستخدم فيشمل البرامج التي يعمل عليها المستخدم، وعادةً فإن برامج المستخدم User programs لا تملك سماحيات بالقراءة والكتابة مباشرة من الطرفيات، ولا الوصول إلى نظام الملفات، أو العتاد، وإنما بإمكانها فقط النفاذ إلى السجلات والذاكرة المخصصة للبرنامج. فإن كان هناك برنامج يريد الوصول إلى جزء ما من العتاد مثل قراءة ملف، وهذا البرنامج موجود في فضاء المستخدم فإنه لا يستطيع أن يصل إلى القرص الصلب Hard disk مباشرة، بل عليه أن يقوم بعمل استدعاء نظام System Call أو TRAP حيث يخبر نظام التشغيل بما يريد (وذلك حسب القيمة المسندة إلى السجل \$v0)، وبالتالي يقوم البرنامج بتسليم الدفعة لنظام التشغيل الذي بدوره يقوم بالتحويل إلى فضاء النواة لإحضار المطلوب ثم العودة مرة أخرى إلى البرنامج. ويوفر برنامج المحاكاة MARS مجموعة من الخدمات services التي يقوم بها نظام التشغيل عن طريق استدعاءات النظام syscall. وفيما يلي جدول يبين بعض استدعاءات النظام المستخدمة:

النتيجة Result	المعاملات Arguments	الاستدعاء SysCall	الخدمة Service
	\$a0 = integer to print	1	Print int
	\$f12 = float to print	2	Print float
	\$f12 = double to print	3	Print double

Print string	4	\$a0=start address of the null-terminated string to print	
Read integer	5		\$v0 contains integer read
Read float	6		\$f0 contains float read
Read double	7		\$f0 contains double read
Read string	8	\$a0 = address of input buffer \$a1 = maximum number of characters to read	
Exit (terminate execution)	10		
Print character	11	\$a0 = character to print	
Read character	12		\$v0 contains character read

كيفية استخدام خدمات النظام SYSCALL:

1- تحميل رقم استدعاء النظام في السجل \$v0.

2- تحميل قيم المعاملات - إن وجدت- في سجلات المعاملات مثل السجلات
\$a0, \$a1 ... إلخ

3- كتابة تعليمة استدعاء النظام وهي SYSCALL.

مثلاً لطباعة العدد 100 على الشاشة:

```
li $v0, 1  
li $a0, 100  
syscall
```

أما لقراءة عدد مدخل من لوحة المفاتيح نكتب:

```
li $v0, 5  
syscall  
# $v0= input value
```

ولطباعة رسالة معينة msg:

```
li $v0, 4  
la $a0, msg  
syscall
```

ولإنهاء البرنامج:

```
li $v0, 10  
syscall
```

أمثلة محلولة:

المثال الأول: برنامج بلغة التجميع MIPS يضع عددين صحيحين في سجلين ثم يجمعهما معاً.

```
## Program to add two plus three
```

```
.text
```

```
.globl main
```

```
main:
```

```
ori    $8,$0,0x2    # put two into register 8
```

```
ori    $9,$0,0x3    # put three into register 9
```

```
add    $10,$8,$9     # add register 8 and 9, put result in 10
```

```
li     $v0,10
```

```
syscall
```

```
## End of file
```

• انسخ تعليمات البرنامج السابق إلى محرر النصوص واحفظه باسم

.addtwo.asm



• قم بتجميع البرنامج بالضغط على الأيقونة .



• بعد الضغط على أيقونة التنفيذ نلاحظ تغير محتوى السجلات

\$8,\$9,\$10 والتي تقابل على التوالي \$t0,\$t1,\$t2 لتصبح 2,3,5 كما هو موضح

في نافذة السجلات في الشكل التالي:

Coproc 0		Coproc 1
Registers		
Name	Number	Value
\$zero	0	0
\$at	1	0
\$v0	2	10
\$v1	3	0
\$a0	4	0
\$a1	5	0
\$a2	6	0
\$a3	7	0
\$t0	8	2
\$t1	9	3
\$t2	10	5
\$t3	11	0
\$t4	12	0

وبالعودة إلى أسطر البرنامج فإن:

- الإشارة # تعني أن ما بعدها عبارة عن تعليق، وبالتالي فإن كل ما هو موجود في السطر على يمينها سيتم تجاهله من قبل المجمع assembler ولا يتم تحويله لأية تعليمة آلة.

- text : وهي عبارة عن توجيه يخبر المجمع بأن الأسطر التالية هي عبارة عن ملف مصدري "text". للبرنامج، ولا يتم تحويل هذه الكلمة إلى أي مقابل بلغة الآلة.

- globl main : وهو توجيه آخر بأن المعرف main سيتم استخدامه (بشكل شمولي globally) خارج الملف المصدري، بمعنى أن عدة ملفات مصدريّة يمكنها استخدام هذا الرمز للإشارة إلى نفس الموضع في مكان التخزين، مع الانتباه إلى أنه يُكتب globl وليس global !.

-السطر main: يعرّف عنواناً رمزياً symbolic address وهو عبارة عن رمز لعنوان موقع في الذاكرة تبدأ عنده تعليمات البرنامج، حيث أن استخدام العنوان الرمزي أسهل بكثير من استخدام العنوان العددي، وبالعنوان الرمزي فإن المبرمج يشير إلى مواقع الذاكرة بالاسم وبدع المجمع assembler يقوم بإيجاد العنوان العددي المقابل.

-الأسطر الفارغة يتم تجاهلها.

-التعليمة الأولى في البرنامج تضع القيمة 2 في السجل \$8 (تُمثل الأعداد على 32 بت).

-والتعليمة الثانية تضع القيمة 3 في السجل \$9 .

-أما التعليمة الأخيرة فهي تجمع محتوى السجلين \$8 و\$9 وتضع الناتج في \$10.

-السطران الأخيران من البرنامج يتضمنان استدعاء نظام System Call للخروج exit من البرنامج، وهذا يقابل 0 return في لغة C/C++.

- بعد تجميع البرنامج إن وجد فيه أخطاء فإن برنامج المحاكاة سيعرض رسائل الأخطاء أسفل الشاشة تحت التبويب MARS messages، وعندها تحتاج لتصحيح الأخطاء في محرر النصوص ، وحفظ التعديلات، ثم إعادة تجميع البرنامج ثم تنفيذه.

- يمكن اختيار تنفيذ البرنامج تعليمة تعليمة وذلك بالضغط على الأيقونة  1 -
أو بالضغط المتكرر على المفتاح F7 لتنفيذ كل تعليمة على حدة، حتى تصل إلى تنفيذ التعليمة ori \$8,\$0,0x2 والتي تقوم بشحن السجل \$8 بالقيمة 2، فنجد في لوحة السجلات تغير قيمة السجل \$8 لتصبح 2، ونلاحظ أن تمثيل هذه التعليمة بلغة الآلة يقابل: 0x34080002 كما هو موضح بجانب التعليمة في مقطع النص، وبالضغط على F7 مرة أخرى ينتقل عداد البرنامج إلى التعليمة التالية ori \$9,\$0,0x3 ونلاحظ أيضاً تغير قيمة السجل \$9 لتصبح 3. ثم بالضغط على F7 مرة أخرى نجد أن السجل \$10 أصبح يحوي حاصل جمع العددين:

$$0x00000002 + 0x00000003 = 0x00000005$$

الشكل التالي يوضح مقطعي النص والمعطيات في الذاكرة، ونلاحظ أن محتوى الذاكرة لم يتم عليه أي تعديل:

Edit		Execute	
Text Segment			
Bkpt	Address	Code	Basic
☐	0x00400000	0x34080002	ori \$8,\$0,0x00000002
☐	0x00400004	0x34090003	ori \$9,\$0,0x00000003
☐	0x00400008	0x01095020	add \$10,\$8,\$9
☐	0x0040000c	0x2402000a	addiu \$2,\$0,0x0000000a
☐	0x00400010	0x0000000c	syscall
Data Segment			
Address	Value (+0)	Value (+4)	Value (+8)
0x10010000	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000

- بعد تجميع البرنامج يمكن اختيار تنفيذ البرنامج من بدايته إلى نقطة توقف break-point عند تعليمة معينة ولكن تعليمة ori \$9,\$0,0x3 بوضع إشارة ✓ في المربع قبل التعليمة، كما بالشكل التالي:

Text Segment			
Bkpt	Address	Code	Basic
☐	0x00400000	0x34080002	ori \$8,\$0,0x00000002
☒	0x00400004	0x34090003	ori \$9,\$0,0x00000003
☐	0x00400008	0x01095020	add \$10,\$8,\$9
☐	0x0040000c	0x2402000a	addiu \$2,\$0,0x0000000a
☐	0x00400010	0x0000000c	syscall

وعند إعطاء أمر تنفيذ البرنامج سيتم تنفيذ التعليمة ori \$8,\$0,0x2 وتتعديل قيمة السجل \$8 لتصبح 2، بينما يتوقف التنفيذ عند التعليمة التالية لها وهي التعليمة: ori \$9,\$0,0x3 وبالتالي لا تنفذ التعليمة ولا تتعدل قيمة السجل \$9 إلى أن يتم الضغط على أيقونة التنفيذ أو أيقونة التنفيذ خطوة خطوة 1.

- يمكن إظهار محتوى السجلات والذاكرة والعناوين بالقيم الست عشرية من خلال قائمة الإعدادات.

- لإنهاء البرنامج نستخدم استدعاء النظام التالي في نهاية البرنامج:

```
li $v0,10
```

```
syscall
```

وفي المعالج الحقيقي عندما يصل المعالج إلى نهاية تنفيذ البرنامج فإنه يعيد التحكم إلى نظام التشغيل.

المثال الثاني: برنامج بلغة التجميع MIPS يطلب من المستخدم إدخال عددين صحيحين فيقوم بجمعهما وطباعة النتيجة على الشاشة.

افتح محرر النصوص وانسخ إليه نص البرنامج التالي ثم احفظه باسم .add2.asm

```
.data
```

```
msg0: .asciiz "Adding numbers:\n"
```

```
msg1: .asciiz "A?"
```

```
msg2: .asciiz "B?"
```

```
result: .asciiz "Sum ="
```

```
newline: .asciiz "\n"
```

```
.text
```

```
.globl main
```

```
main:
```

```
# Print "Adding numbers\n"
```

```
la $a0, msg0
```

```
li $v0, 4
```

```

        syscall

#Print "A?"

        la $a0, msg1
        li $v0, 4
        syscall

#Get the value for A (store in $v0)

        li $v0, 5
        syscall

#Move $v0 to $t0: $t0 = 0 + $v0
        add $t0, $zero, $v0

#Print "B?"

        la $a0, msg2
        li $v0, 4
        syscall

#Get the value for B (store in $v0)

        li $v0, 5
        syscall

#Add A and B: $t0 = $t0 + $v0
        add $t0, $t0, $v0

#Print "The sum is"

        la $a0, result

```

```

li $v0, 4

syscall

#Print result ($t0 is the result)

add $a0, $zero, $t0

li $v0, 1

syscall

#Print a new line

la $a0, newline

li $v0, 4

syscall

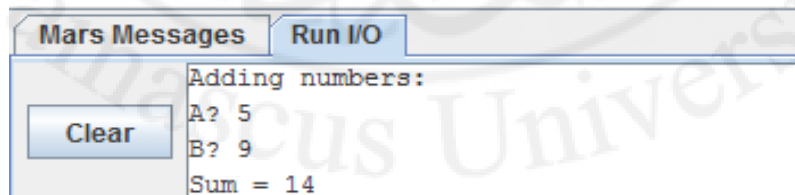
#Terminate the program

li $v0, 10

syscall

```

بعد تجميع الملف add2.asm وإعطاء أمر التنفيذ Run نلاحظ في نافذة الدخل/الخرج رسائل تطلب من المستخدم إدخال قيمة كل من العددين الصحيحين A,B وبعدها تتم طباعة ناتج جمعهما كما في الشكل التالي:



وبالعودة إلى أسطر البرنامج فإن: التوجيه data. يعني بداية مقطع ذاكرة المعطيات، التوجيه word. يعني: ضع عدداً صحيحاً (مؤلفاً من 32 بت وبطريقة

الإتمام الثنائي) هنا. وبشكل افتراضي فإن التعبير عن الأعداد الصحيحة يتم حسب التمثيل العشري base 10، وبالتالي عند كتابة: word 17. فإن المجمع يحول العدد $10(17)$ إلى المكافئ الثنائي، أما إذا أردت إعطاء القيم بالتمثيل الست عشري hex فيمكن كتابة word 0x11. والتي تكافئ word 17..

يتم التصريح عن المتحولات المستخدمة في البرنامج "variables" تحت السطر data. حيث أن المتحولات مخزنة في ذاكرة المعطيات، مع العلم أن العمليات الحسابية والمنطقية في معالجات MIPS يمكن انجازها باستخدام السجلات فقط، وهذا يعني أنه لا يمكن تغيير قيم المتحولات مباشرة بل لا بد من نقل هذه القيم من الذاكرة إلى سجلات المعالج لإجراء العمليات المطلوبة. ويتم التصريح عنها كما يلي:

<variableName>: <dataType> <initialValue>

والجدول التالي يبين أمثلة عن بعض أنماط المعطيات :

اسم المتحول	نمط المعطيات	القيمة الابتدائية	ملاحظات
var1:	.word	12	# A word consists of 4 bytes, the same as int var1=12 in C++
array1:	.word	1,2,3,4,5	# same as int a[5]={1,2,3,4,5} in C++, reserve space for words

			and store 1,2,3,4,5 in them
Array2:	.word	2 4 6	# same as int a[3] = {2,4,6} in C++
Array3:	.word	0:6	# same as int a[6]={0,0,0,0,0,0} in C++
Str1:	.byte .byte .byte	0x32 # '2' 0x4a # 'J' 0 # '\0' in ASCII	# String type is array of char, each char is a byte, same as char Str1[3]='2','J','\0' in C++
Str2:	.asciiz	"2J"	# NULL terminated string Equivalent to Str1
Str3:	.ascii	"2J"	# insert Ascii string into next few bytes of memory, without null byte at end
	.space n		# reserve space for n bytes of memory

المثال الثالث: فيما يلي مثال يوضح كيف يمكن تخزين المعطيات في مقطع المعطيات .data :

.data

X: .word 1 2 3 # like int X[3] = {1,2,3} ; in C++

String1: .ascii "AB" # same as string String1 = "AB"

Y: .byte 67 68 0 # This is the same as string "CD" ,
0 is treated as end of string symbol

.text

main:

li \$v0,10

syscall

عند تنفيذ البرنامج واستعراض محتوى ذاكرة المعطيات نجد فيها أماكن توضع المتحولات كالتالي:

X				X + 4		
1				2		
String1	String1+1	String1+2	String1+3	String1+4	..	..
65	66	0	67	68	0	..

والشكل التالي يوضح توضع المعطيات السابقة في مقطع ذاكرة المعطيات:

Data Segment						
Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)
0x10010000	0x00000001	0x00000002	0x00000003	0x43004241	0x00000044	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010010	1	2	3	C	B	3

وفي حال تفعيل خيار ASCII في هذه النافذة تظهر الأحرف A,B,C,D في

الذاكرة بدل القيم السابقة وهي التمثيل الست عشري للأحرف حسب ASCII:

Data Segment						
Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)
0x10010000	\0 \0 \0 .	\0 \0 \0 .	\0 \0 \0 .	C \0 B A	\0 \0 \0 D	\0 \0 \0 \0
0x10010020	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0

نلاحظ من الشكلين السابقين بأن كل متحول من نمط المعطيات word يحجز 4 بايت، وبأنه يتم تخزين العناصر من مصفوفة الكلمات word array بشكل متعاقب، وأن النمط "asciiz" يكافئ مصفوفة من نمط byte (أو char في لغة C++)، وأن قيمة ASCII لكل محرف يتم تخزينها في الذاكرة بشكل متعاقب.

وفيما يلي جدول يبين تمثيل المحارف باستخدام ترميز ASCII، كل محرف مرمز على 8 بت. ويظهر الجدول القيم بالنظام الست عشري hex. رأس العمود يعطي الخانة الدنيا من القيمة ورأس السطر يعطي الخانة العليا من القيمة ممثليين بالترميز hex. فمثلاً المحرف 'K' نجد أن رأس العمود له هو B ورأس السطر له هو 4 وبالتالي تمثيل القيمة 'K' في ترميز ASCII هو 0x4B أو ثنائياً 0100 1011. وبالمثل فإن تمثيل القيمة 'A' في ترميز ASCII هو 0x41 وعشرياً 65. وقد تم كتابة محارف التحكم بأحرف كبيرة مثل: SP(space)، BS (backspace)، carriage return)، وعملياتاً فالمسافة SP حرف مطبوع وليس محرف تحكم.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2	SP	!	"	#	\$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	A	b	c	d	e	f	g	h	i	j	k	l	m	n	O
7	p	Q	r	s	t	u	v	w	x	y	z	{		}	~	DEL

المثال الرابع: برنامج بلغة التجميع MIPS لطباعة العبارة Hello World:

<pre> #----- Data Segment ----- .data # declare the string msg msg: .asciiz "Hello World\n" #----- Text Segment ----- .text # main is the default starting point of a MIP program main: # Execute the "print_str" syscall li \$v0, 4 la \$a0, msg syscall # Execute the "exit" system call li \$v0, 10 syscall </pre>	<pre> // C++ version // declare the string msg char msg[] = {'H','e','l','l','o',' ','\n','W','o','r','l','d','\n','\0'} // main is the default starting point of the program void main() { cout << msg; exit(); } </pre>
--	--

قم أولاً باختيار التبويب Edit، ثم اختر من قائمة File إنشاء ملف جديد

New، اختر اسم الملف مثلاً hello.asm.

اكتب تعليمات MIPS السابقة (في العمود الأيسر من الجدول) واحفظ هذه التعديلات من القائمة File الخيار Save. وهذا توضيح سريع لعمل التعليمات:

السطر 2: يتم إعداد سلسلة محارف متوضعة في مقطع المعطيات Data.

الأسطر 4،5،6 : تعليمات لطباعة سلسلة المحارف السابقة.

السطران 7،8 : تعليمات يستخدمها البرنامج للخروج من البرنامج.

اضغط على أيقونة التجميع Assemble، وإن كان هناك أي خطأ مطبعي فستظهر رسائل خطأ في تبويب الرسائل أسفل الشاشة، وعندها يمكنك تصحيح الخطأ وإعادة الضغط على الأيقونة Assemble، عندها يتم عرض تبويب التنفيذ Execute، ونلاحظ تحت هذا التبويب:

مقطع النص Text segment وفيه:

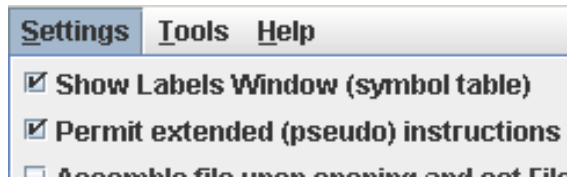
العمود Address: يحوي عنوان الذاكرة بالتمثيل Hex.

العمود Code: وفيه التعليمات بلغة الآلة بالتمثيل Hex.

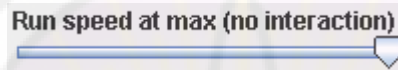
العمود Source: وفيه التعليمات بلغة التجميع والتي تم كتابتها في المحرر.

نلاحظ أن التعليمة la هي تعليمة زائفة pseudo، فيقوم المجمع بتوليد تعليمتين حقيقيتين بدلاً عنها لوضع العنوان المذكور في سجل، ونجد في العمود Basic التعليمات الحقيقية للبرنامج مع استخدام أرقام السجلات بدلاً من أسمائها، وقيم عناوين الذاكرة بدلاً من التسميات.

في قائمة Settings قم بتفعيل الخيار Show Labels Window لرؤية نافذة التسميات Labels والتي تعرض العناوين المقابلة للتسميات، مثلاً عنوان msg هو 0x10010000.

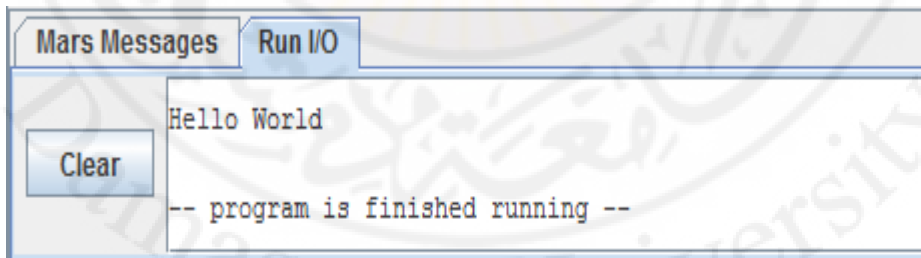


يمكن استخدام شريط التمرير لتغيير سرعة التنفيذ إلى 10 تعليمات بالثانية (مثلاً)، للتمكن من مراقبة التنفيذ بدلاً من انتهائه مباشرة خاصة في البرامج التي تأخذ وقتاً طويلاً عند التنفيذ.



في نافذة مقطع المعطيات Data قم بتفعيل الخيار ASCII وعندها ترى محارف العبارة Hello World بترتيب معين، تبدأ من العنوان 0x10010000، وإذا ألغيت تفعيل الخيار ASCII سيظهر محتوى مقطع المعطيات على شكل كلمات بالتمثيل الست عشري Hex.

بعد الضغط على أيقونة التنفيذ Run تجد نتيجة التنفيذ في أسفل النافذة تحت التبويب Run I/O كما يلي:



طريقة طباعة سلسلة المحارف (string): في هذا المثال نجد أن توضع سلسلة المحارف في الذاكرة سيكون مشابهاً لما يلي:

msg	msg	msg	msg	msg	msg	..	..	..	..	..	msg	msg
	+1	+2	+3	+4	+5						+11	+12
'H'	'e'	'l'	'l'	'o'	' '	'W'	'o'	'r'	'l'	'd'	'\n'	'\0'

عند تنفيذ التعليمة msg, \$a0 لا فإن عنوان بداية سلسلة المحارف يُسند إلى السجل \$a0، وبالتالي فإن كان (character 'H') msg يتوضع في البايت رقم 1001 من الذاكرة فإن \$a0 = 1001، وبعد إسناد القيمة 4 إلى السجل v0 وتنفيذ استدعاء نظام syscall فإن المعالج عندئذ يعلم أننا نريد طباعة سلسلة المحارف على الشاشة.

يقرأ المعالج بايت بايت من الذاكرة بدءاً من العنوان \$a0 أي 1001 ثم 1002 ثم 1003 وهكذا، ويظهر المحرف المقابل واحداً واحداً حتى الوصول إلى محرف انتهاء سلسلة المحارف وهو ('\0')، ولاحظ أن قيمة ASCII لنهاية سلسلة المحارف هي 0. والشكل التالي يبين مقطع المعطيات في الذاكرة بعد تنفيذ البرنامج:

Data Segment				
Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)
0x10010000	1 1 e H	o W o	\n d l r	\0 \0 \0 \0
0x10010020	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x10010040	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x10010060	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0

نلاحظ أن الأحرف قد خزنت في الذاكرة بشكل معكوس ضمن كل كلمة مؤلفة من 4 بايت، وهذا أمر خاص بالتصميم العتادي وتسمى طريقة التخزين في هذه الحالة بالتخزين النهوي الصغير little endian ويعني أنه يتم تخزين البايتات بحيث يوضع البايت الأقل دلالة في البداية، بينما في النهوي الكبير Big endian يوضع البايت الأكثر دلالة في البداية لذا تظهر سلسلة المحارف في الذاكرة بالترتيب الذي تمت كتابتها به.

المثال الخامس: برنامج لإيجاد أول 12 عنصر من سلسلة فيبوناتشي بلغة

```
.data
fibs: .word 0:12 # "array" of 12 words to contain fib values
size: .word 12 # size of "array "

.text # Compute first twelve Fibonacci numbers and put
      # in array, then print

la $t0, fibs # load address of array
la $t5, size # load address of size variable
lw $t5, 0($t5) # load array size
li $t2, 1 # 1 is first and second Fib. number
sw $t2, 0($t0) # F[0] = 1
sw $t2, 4($t0) # F[1] = F[0] = 1
addi $t1, $t5, -2 # Counter for loop, execute (size-2) times
loop: lw $t3, 0($t0) # Get value from array F[n]
      lw $t4, 4($t0) # Get value from array F[n+1]
      add $t2, $t3, $t4 # $t2 = F[n] + F[n+1]
      sw $t2, 8($t0) # Store F[n+2] = F[n] + F[n+1] in array
      addi $t0, $t0, 4 # increment address of Fib.
      addi $t1, $t1, -1 # decrement loop counter
      bgt $t1, $zero, loop # repeat if not finished yet.
la $a0, fibs # first argument for print (array)
```

```

add $a1, $zero, $t5 # second argument for print (size)

jal print           # call print routine .

li $v0, 10          # system call for exit

syscall             # we are out of here.

##### routine to print the numbers on one line .

.data
space: .asciiz " "    # space to insert between numbers
head: .asciiz "The Fibonacci numbers are:\n"

.text
print: add $t0, $zero, $a0 # starting address of array
      add $t1, $zero, $a1 # initialize loop counter to array size
      la $a0, head        # load address of print heading
      li $v0, 4            # specify Print String service
      syscall             # print heading
out: lw $a0, 0($t0)        # load fibonacci number for syscall
      li $v0, 1            # specify Print Integer service
      syscall             # print fibonacci number
      la $a0, space        # load address of spacer for syscall
      li $v0, 4            # specify Print String service
      syscall             # output string
      addi $t0, $t0, 4     # increment address

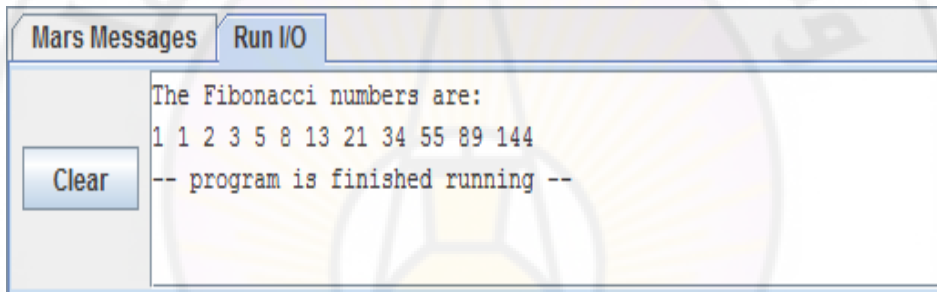
```

```
addi $t1, $t1, -1    # decrement loop counter
```

```
bgt $t1, $zero, out  # repeat if not finished
```

```
jr $ra               # return
```

نسخ تعليمات البرنامج السابق إلى محرر النصوص واحفظه باسم Fibonacci.asm. ثم قم بتجميع البرنامج بالضغط على الأيقونة  وعند الضغط على أيقونة التنفيذ  تظهر قيم سلسلة فيبوناتشي في مقطع المعطيات. ونلاحظ الخرج:



والشكل التالي يظهر محتويات مقطع المعطيات في الذاكرة بعد تنفيذ البرنامج:

Data Segment									
Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)	
0x10010000	1	1	2	3	5	8	13	21	
0x10010020	34	55	89	144	12	1750335520	1766203493	1634627426	
0x10010040	543777635	1651340654	544436837	979726945	10	0	0	0	
0x10010060	0	0	0	0	0	0	0	0	
0x10010080	0	0	0	0	0	0	0	0	
0x100100a0	0	0	0	0	0	0	0	0	
0x100100c0	0	0	0	0	0	0	0	0	
0x100100e0	0	0	0	0	0	0	0	0	

At the bottom of the window, there are controls for the data segment:

- Buttons:  
- Dropdown menu: 0x10010000 (.data)
- Checkboxes: ☒ Hexadecimal Addresses, ☐ Hexadecimal Values, ☐ ASCII

نلاحظ أن عناصر سلسلة فيبوناتشي مخزنة في الذاكرة بدءاً من العنوان 0x10010000 وحتى العنوان 0x1001002c. وقبل التنفيذ كانت قيمها الابتدائية

أصفاً. أما عدد عناصر السلسلة size فمخزن في الذاكرة عند العنوان 0x10010030. بينما عناوين الذاكرة التالية تحوي سلاسل محرفية strings.

المثال السادس: برنامج بلغة التجميع MIPS يقوم باستدعاء تابع فيبوناتشي Fibonacci من البرنامج الرئيسي main.

.data

#Data for prompts and output description

prmt1: .asciiz "\nThis program computes the Fibonacci function."

prmt2: .asciiz "\nEnter value for n:"

result: .asciiz "fib(n) ="

.text

main:

Print the prompts

li \$v0, 4 # print_str system service...

la \$a0, prmt1 # passing address of first prompt

syscall

li \$v0, 4 # print_str system service...

la \$a0, prmt2 # passing address of 2nd prompt syscall

syscall

#Read n and call fib with result

li \$v0, 5 # read_int system service

```

syscall

move $a2, $v0 # $a2 = n = result of read

li $a1, 0      # $a1 = fib(0)
li $a0, 1      # $a0 = fib(1)
jal fib        # call fib(n)
move $s1,$v0   # $s1 = fib(n)

#Print result

li $v0,4       # print_str system service...
la $a0, result # passing address of result
syscall

li $v0,1 # print_int system service...
move $a0,$s1 # ... passing argument fib(n)
syscall

#Call system – exit

li $v0, 10
syscall

#-----#

#Algorithm for Fib(a, b, count):
fib :

    bne $a2,$zero, fibne0 # if count == 0...
    move $v0,$a1 # ... return b

```


jr \$31

```
fibne0:                # Assert: n != 0

    addi $a2, $a2, -1    # count = count - 1

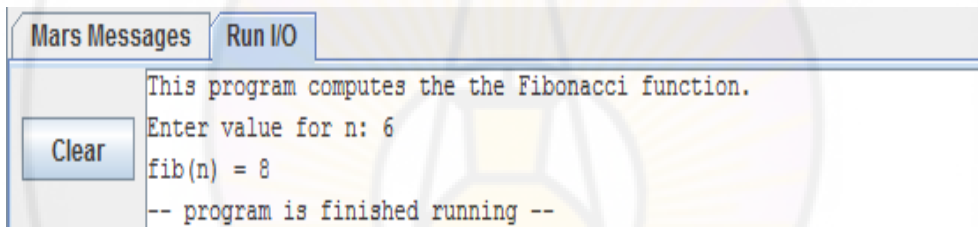
    add $t1, $a0, $a1    # $t1 = a + b

    move $a1, $a0        # b = a

    move $a0, $t1        # a = a + old b

    j fib
```

بتنفيذ البرنامج نحصل على ما يلي:



المثال السابع: برنامج بلغة التجميع MIPS لحساب عاملي عدد $fact(n)$ بالطريقة التكرارية.

.data

enterN: .asciiz "Please enter the n value: \n"

output: .asciiz "Result is: "

.text

j main

factorial:

iterative factorial procedure

\$a0 : number, \$v0 : factorial of the number

addi \$sp, \$sp, -4

sw \$ra, 0(\$sp)

li \$v0, 1

li \$s0, 1

factorial_begin:

beq \$s0, \$a0, factorial_end # n == 1?

mul \$v0, \$v0, \$a0 # \$v0 = \$v0 * n

addi \$a0, \$a0, -1 # n = n - 1

j factorial_begin

factorial_end:

lw \$ra, 0(\$sp)

addi \$sp, \$sp, 4

jr \$ra

main:

la \$a0, enterN #Ask for the first param, n.

li \$v0, 4 #String syscall

syscall #Prints out string.

li \$v0, 5

syscall #Places inputted value in v0.

```
move $a0, $v0    # $a0 = n
```

```
jal factorial
```

```
move $s0,$v0
```

```
la $a0, output
```

```
li $v0,4
```

```
syscall
```

```
move $a0, $s0
```

```
li $v0,1
```

```
syscall
```

```
li $v0,10
```

```
syscall
```

وعند التنفيذ نحصل على ما يلي:

Mars Messages	Run I/O
Clear	Please enter the n value:
	4
	Result is: 24
	-- program is finished running --

المثال الثامن: برنامج بلغة التجميع MIPS لحساب عاملي عدد $\text{fact}(n)$ بالطريقة العودية.

This program computes factorial of entered number with
recursion . it illustrates how to set up the procedure stack.

```

.data
    msg_str: .asciiz "Enter some Number:"

.text

.globl main
main:
    la $a0, msg_str
    li $v0, 4
    syscall
    li $v0, 5
    syscall
    move $a0,$v0
    jal fac
    move $a0,$v0           #get result
    li $v0,1               #print integer
    syscall
    li $v0,10
    syscall

# fac(arg) – computes factorial of arg (arg!),
# argument is passed in $a0

#prologue to procedure
fac:

```

```

addi $sp,$sp,-8           #push space

sw  $s0,0($sp)            # save $s0, which we use

sw  $ra,4($sp)            # save return address

#start of actual procedure work

move $s0,$a0              #get argument ($a0)

li  $v0,0x00000001 # 1

beq $s0,$v0,L2            #end of recursion?

addi $a0,$s0,-1           # set up argument (f-1)

jal fac                   #recursive call

mult $v0,$s0              #multiply

mflo $v0                  #return mul result

j L3                      #exit procedure via epilogue

L2:

li  $v0,0x00000001        # return value

# epilogue to exit procedure

L3:

lw  $ra,4($sp)            # restore $ra

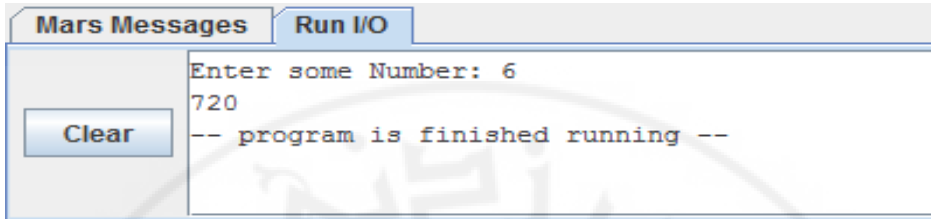
lw  $s0,0($sp)            # restore $s0

addi $sp,$sp,8            # pop activation frame

jr  $ra                   # return

```

بتنفيذ البرنامج نحصل على ما يلي:



المثال التاسع: برنامج لطباعة عدد ذي فاصلة عائمة باستخدام سجلات وتعليمات أعداد الفاصلة العائمة.

.data

fp: .float 1.25

.text

la \$t1, fp

l.s \$f0, (\$t1)

li \$v0, 10

syscall

عند تنفيذ البرنامج يعطي محتوى السجل \$f0 كما يلي:

Registers	Coproc 1	Cop
Name	Float	
\$f0	0x3fa00000	
\$f1	0x00000000	
\$f2	0x00000000	

حيث أن العدد 1.25 يُمثل حسب المعيار IEEE754 كما يلي:

0 0111 1111 010 0000 0000 0000 0000 0000

= 0 x 3fa0 0000

المثال العاشر: برنامج لإجراء عمليات حسابية على أعداد ذات فاصلة عائمة.

.data

a: .float 1.5

b: .float 0.125

c: .float 3.75

.text

la \$t0,a # [lui \$1,0x1001]

lwc1 \$f0, 0(\$t0)

l.s \$f1,b # [lui \$1,0x1001 , lwc1 \$f1, 4(\$t0)]

l.s \$f2,c # [lui \$1,0x1001 , lwc1 \$f2, 8(\$t0)]

add.s \$f3,\$f0,\$f1

sub.s \$f4, \$f2, \$f1

sub.s \$f5, \$f0, \$f2

mul.s \$f6,\$f0,\$f0

div.s \$f7, \$f2, \$f0

swc1 \$f7, 12(\$t0) # store \$f7 in mem[\$t0+12]

mov.s \$f12, \$f6

li \$v0,2

syscall # to print the content of \$f12 .

li \$v0,10

syscall # to exit.

نلاحظ بالتنفيذ أن قيم سجلات الأعداد ذات الفاصلة العائمة أصبحت كما يلي:

Registers		Coproc 1	Registers		Coproc
Name	Float		Name	Float	
\$f0	1.5		\$f0	0x3fc00000	
\$f1	0.125		\$f1	0x3e000000	
\$f2	3.75		\$f2	0x40700000	
\$f3	1.625		\$f3	0x3fd00000	
\$f4	3.625		\$f4	0x40680000	
\$f5	-2.25		\$f5	0xc0100000	
\$f6	2.25		\$f6	0x40100000	
\$f7	2.5		\$f7	0x40200000	
\$f8	0.0		\$f8	0x00000000	
\$f9	0.0		\$f9	0x00000000	
\$f10	0.0		\$f10	0x00000000	
\$f11	0.0		\$f11	0x00000000	
\$f12	2.25		\$f12	0x40100000	

والقيم المخزنة في الذاكرة هي:

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)
0x10010000	0x3fc00000	0x3e000000	0x40700000	0x40200000

والقيمة المطبوعة على الشاشة:

Mars Messages	Run I/O
2.25	-- program is finished running --

المثال الحادي عشر: برنامج لحساب مساحة الدائرة $area = \pi * r * r$ ، باستخدام سجلات الأعداد ذات الفاصلة العائمة مع طباعة النتيجة.

. data

msg1: .asciiz " Enter the value of radius (r = ?) \n"

msg2: .asciiz "area= pi*r*r= "


```
pi: .double 3.141592653589793
```

```
.text
```

```
la $a0, msg1
```

```
li $v0, 4
```

```
syscall      # print msg1
```

```
li $v0, 7     # read_double ( r )
```

```
syscall      # radius <- user input (in $f0 == $f0,$f1)
```

```
la  $a0, pi
```

```
l.d  $f12, 0($a0)      # a := pi
```

```
mul.d  $f12, $f12, $f0  # a := a * r
```

```
mul.d  $f12, $f12, $f0  # a := a * r
```

```
la $a0, msg2
```

```
li $v0, 4
```

```
syscall      # print msg2
```

```
li $v0, 3     # print_double
```

```
syscall      # print area
```

```
li $v0, 10
```

```
syscall
```

Mars Messages	Run I/O
Enter the value of radius (r = ?)	
Clear	3.15
	area= pi*r*r = 31.172453105244717
	-- program is finished running --

تمارين غير محلولة

التمرين (1): ما هو النص المطبوع على الشاشة بعد تنفيذ البرنامج التالي؟

علل.

.data

```
str1: .byte 65 66 # declare the string mesgs
```

```
mesg: .asciiz "Hello World\n"
```

.text

```
li $v0, 4
```

```
la $a0, str1
```

```
li $t0, 1
```

```
add $a0, $a0, $t0
```

```
syscall
```

```
li $v0, 10
```

```
syscall
```

التمرين (2): اكتب برنامجاً بلغة التجميع MIPS يقوم المستخدم بإدخال مصفوفة أعداد صحيحة (بطول محدد) فيقوم البرنامج بإرجاع عدد الأعداد الموجبة وعدد الأعداد السالبة، وعدد الأعداد الفردية وعدد الأعداد الزوجية.

التمرين (3): اكتب برنامجاً بلغة التجميع MIPS يقوم المستخدم بإدخال مصفوفة أعداد صحيحة، فيقوم البرنامج بإرجاع مصفوفة كل عنصر فيها هو مربع العنصر المقابل من المصفوفة المدخلة (أي العنصر الأول المرجع = مربع العنصر الأول المدخل).

التمرين (4): اكتب برنامجاً بلغة التجميع MIPS يقوم المستخدم بإدخال عددين، فيقوم البرنامج باستدعاء تابع يقوم بإيجاد مضاعفات العدد 3 المحصورة بين هذين العددين المدخلين، ثم طباعتها، فمثلاً إذا أدخل المستخدم العددين 7 و 22 فيتم طباعة الأعداد التالية: (9، 12، 15، 18، 21).

التمرين (5): اكتب برنامجاً بلغة التجميع MIPS لحساب قوة عدد x^y بالطريقة التكرارية.

التمرين (6): اكتب برنامجاً بلغة التجميع MIPS لحساب قوة عدد x^y بالطريقة العودية.

التمرين (7): اكتب التعليقات الموضحة لتعليمات البرنامج التالي، ثم استنتج عمل البرنامج وما هي النتيجة التي سيتم طباعتها؟

ثم تأكد من الحل عن طريق تنفيذه باستخدام برنامج المحاكاة MARS:

.text

main:

la \$a0, array1

addi \$a1, \$zero, 8

jal x

j finish

x: addi \$t7, \$zero, 0

addi \$t2, \$zero, 0

x_loop:

beq \$t7, \$a1, x_finish

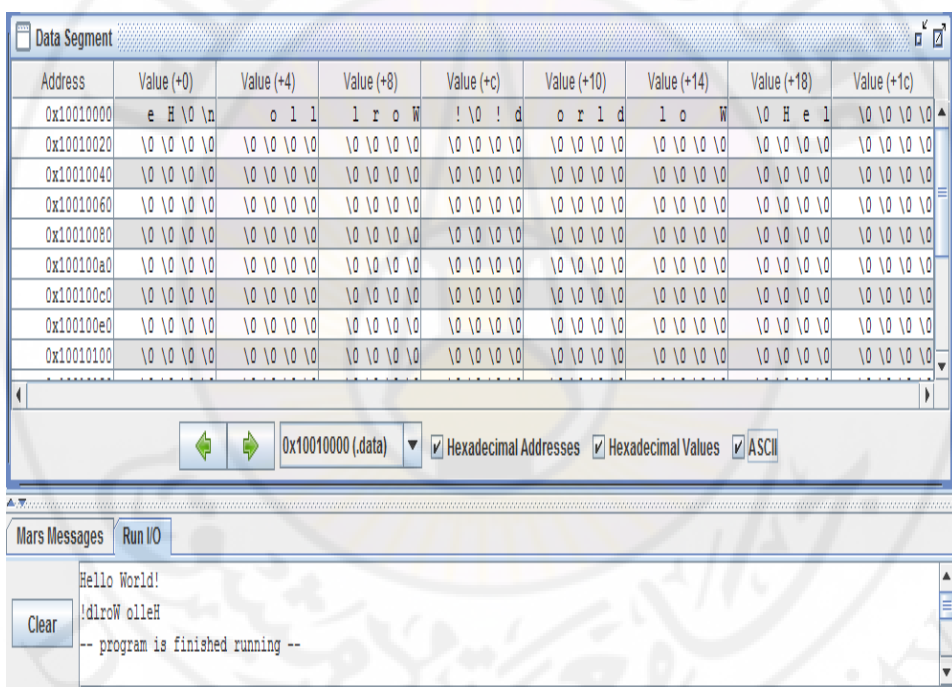
```

lw $t3, 0($a0)
slt $t1, $t3, $t2
bne $t1, $zero, notmore
addi $t2, $t3, 0
notmore:
    addi $t7, $t7, 1
    addi $a0,$a0, 4
    j x_loop
finish:
    jr $ra
x_finish:
    addi $a2, $t2, 0
    move $a0, $a2
    li $v0, 1
    syscall
exit:
    li $v0,10
    syscall
.data
array1: .word 3,115,9,10,80,6,4

```

التمرين (8): اكتب برنامجاً بلغة التجميع MIPS يقوم المستخدم بإدخال سلسلة محارف، فيقوم البرنامج بحفظها في الذاكرة واستدعاء إجرائية تقوم بعكس ترتيب محارف السلسلة وحفظها في موقع آخر من الذاكرة بالشكل المعكوس، وطباعة السلسلة المدخلة والسلسلة المعكوسة.

مثلاً إذا قام المستخدم بإدخال العبارة: "Hello World!" فينتج عند التنفيذ ما يلي:



التمرين (9): اكتب برنامجاً بلغة التجميع MIPS يتيح للمستخدم الخيارات التالية:

- 1- تحويل درجة الحرارة من سيلزيوس إلى فهرنهايت
- 2- تحويل درجة الحرارة من فهرنهايت الى سيلزيوس

ثم يقوم المستخدم بإدخال الخيار ودرجة حرارة ما، لتحويلها من وحدة قياس إلى أخرى حسب رقم الخيار المدخل.

علماً أن قانوني التحويل بين المقياسين كما يلي:

للتحويل من سيلزيوس إلى فهرنهايت: $F = 1.8 \times C + 32$

للتحويل من فهرنهايت إلى سيلزيوس: $C = (F - 32) \times 0.555555$

مستخدماً تعليمات وسجلات الأعداد ذات الفاصلة العائمة. بحيث تحصل عند التنفيذ على نتيجة مشابهة لما يلي:

Mars Messages	Run I/O
<pre>Press 1 to convert Celcius into Farenheit Press 2 to convert Farenheit into Celcius Enter your choice: 1 Enter Value: 36 Result is: 96.799995 -- program is finished running -- Press 1 to convert Celcius into Farenheit Press 2 to convert Farenheit into Celcius Enter your choice: 2 Enter Value: 96 Result is: 35.55552 -- program is finished running --</pre>	

مسرد المصطلحات

Performance	أداء
Exponent	أس
Biased exponent	أس منحاز
system calls	استدعاءات النظام
Sign	إشارة
Frame	إطار
Setting	إعدادات
Chip Select	انتقاء الرقاقة
Bias	انحياز
Offset	انزياح
Stop	إيقاف
User programs	برامج المستخدم
Simulator	محاكٍ - برنامج محاكاة
Globally	بشكل شمولي
Enable	تأهيل
Ignore overflow	تجاهل الطفح
Edit	تحرير

Enhancement	تحسين
Decode	تفكيك
Load	تحميل
Undo	تراجع
Frequency	تردد المعالج
Speedup	تسريع
Label	تسمية
Expression	تعبير
pseudo instruction	تعليلة زائفة
Instruction	تعليلة
Branch	تفرع
Representation	تمثيل
Zero extending	تمديد بالأصفار
sign extending	تمديد بالإشارة
Run	تنفيذ
Function	تابع
Memory Expansion	توسيع الذاكرة
Expanding memory length	توسيع طولي للذاكرة

Expanding memory width	توسيع عرضي للذاكرة
Pause	توقف مؤقت
Octal	ثماني
Binary	ثنائي
Fraction	جزء كسري
Fetch	جلب
Concatenation	جنباً إلى جنب
Service	خدمة
Exit	خروج
Single-Step	خطوة واحدة
Cells	خلايا
double-precision	دقة مضاعفة
single-precision	دقة مفردة
cycle per instruction	دورة بالتعليمية
Instruction Memory	ذاكرة التعليمات
Data Memory	ذاكرة المعطيات
Clock	ساعة
base register	سجل قاعدي

Saved registers	سجلات محفوظة
Temporary registers	سجلات مؤقتة
Hexadecimal	ست عشري
Capacity	سعة
Hardware	عتاد
program counter	عداد البرنامج
Cycle Count	عدد الدورات
Decimal	عشري
Return address	عنوان العودة
symbolic address	عنوان رمزي
Underflow	غيض
Unique	فريد
address space	فضاء العنونة
User Space	فضاء المستخدم
Kernel Space	فضاء النواة
Overflow	فيض
Hard disk	قرص صلب
Jump	قفز

reserved bitpattern	قيم -شكل نمطي من البتات- محجوزة
Words	كلمات
Machine Language	لغة الآلة
Assembly Language	لغة التجميع
Variables	متحولات
local variables	متحولات محلية
compiler	مترجم
Two's Complement	متمم إلى اثنين / متمم ثنائي
One's Complement	متمم إلى واحد / متمم أحادي
Assembler	مجمع
Character (char)	محرّف
Help	مساعدة
compiler designers	مصممو المترجمات
Sign-Magnitude	مطال (طويلة) وإشارة
Data	معطيات
Stack	مكدس
Register File	ملف السجلات
sign/zero extender	ممدد بالإشارة أو بالأصفار

Data Path	مسار المعطيات
Single Cycle Data path	مسار المعطيات أحادي الدور
Multi Cycle Data path	مسار المعطيات متعدد الأدوار
Unsigned	بدون إشارة
Stack pointer	مؤشر المكس (الكدسة)
Global Pointer	مؤشر شمولي
Address Bus	مسرى العناوين
Data Bus	مسرى المعطيات
Result	نتيجة
Text	نص
System	نظام
operating system	نظام التشغيل
Breakpoints	نقاط التوقف
Type	نمط - نوع
little endian	نهوي صغير
Big endian	نهوي كبير
OS kernel	نواة نظام التشغيل
Arguments	معاملات / محدّدات

Runtime	وقت التنفيذ
Reset	يستبدئ -يعيد إلى القيم الابتدائية





المراجع

المراجع العربية:

- 1- د. محمد نوار العوا - د. جورج صنيح- د. فيصل العباس، "تنظيم الحاسوب وبنائه" - دمشق 2002
- 2- معجم المصطلحات المعلوماتية، من منشورات الجمعية العلمية السورية للمعلوماتية - الطبعة الأولى 2000

المراجع الأجنبية:

- 1- David Money Harris; Sarah L. Harris, Digital Design and Computer Architecture, Second Edition (2013)
- 2-William Stallings, Computer Organization And Architecture, Designing For Performance, Eighth Edition, Prentice Hall , Upper Saddle River.
- 3-David A. Patterson; John L. Hennessy, Computer Organization and Design: The Hardware/Software Interface, Fifth Edition (2014)
- 4- Steven T. Karris , Digital Circuit Analysis and Design with Simulink® Modeling and Introduction to CPLDs and FPGAs, Second Edition.
- 5- Charles W. Kann, Introduction To MIPS Assembly Language Programming, 2015, Gettysburg College.

اللجنة العلمية:

د. إياد الخياط

د. عماد الدين محمد

د. مازن المحاييري

المدقق اللغوي:

د. فاطمة تجور

حقوق الطبع والترجمة والنشر محفوظة لمديرية الكتب والمطبوعات