الفصل الرابع
مجموعة التعليمات وأنماط العنونة

1- مقدمة

يتضمن هذا الفصل المفاهيم الأساسية لمجموعة تعليمات الحاسوب وطرق الوصول إلى المعطيات في الذاكرة (أنماط العنونة) ومن أجل ذلك نبدأ بتوضيح بعض المفاهيم الأساسية للذاكرة وعنونتها ونختتم الفصل بكتابة بعض البرامج التي تشرح استخدام مجموعة التعليمات وكذلك أثر استخدام أنماط عنونة مختلفة .

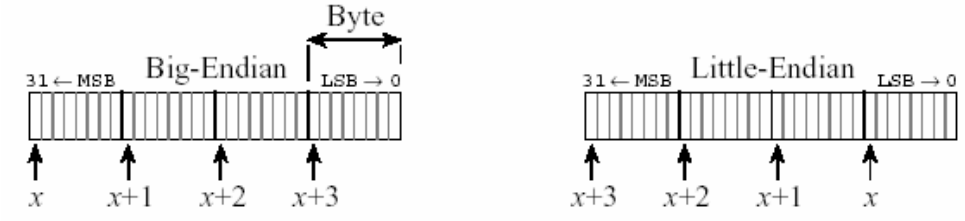
2- العمليات على الذاكرة وتمثيل المعطيات فيها .

تتكون الذاكرة الرئيسية من ملايين الخلايا كل منها تستطيع تخزين بت ثنائية لها القيمة 0 أو 1 ، وتنظم هذه الخلايا بمجموعات من البتات ويمكن أن تحتوي المجموعة بت واحدة او أربع بتات (nipple) أو ثمان بتات (byte) أو ما يسمى كلمة ذاكرة . بقيت هذه التسميات (bit, nipple,byte) متداولة وتعبر عن طول محدد للمجموعة بغض النظر عن التطور ما عدا مفهوم كلمة الذاكرة فيتغير طولها حسب بنية الحاسوب ويكون الطول النموذجي للكلمة 16,32,64,128 bit يبين الشكل 1 الحجوم الشائعة لموقع ذاكرة . وتمثل الكلمة اصغر مجموعة من البتات يمكن تخزينها أو استرجاعها بعملية نفاذ واحدة إلى الذاكرة ، مثلاً في الحواسيب التي تعنون البايت تكون البايت أصغر مجموعة يمكن عنونتها (النفاذ إليها قراءة أو كتابة) ومع ذلك توجد تعليمات تستطيع أن تعنون كلمة مكونة من عدد من البايتات المخزنة بشكل متسلسل (كلمة متعددة البايتات Multi-Byte Word) وتملك معظم الحواسيب الحديثة تعليمات تعنون بايت و بايتين (نصف كلمة) وكلمة (32 bit) أو كلمة مضـاعفة 64 bit).يبين الشكل2 طريقتين وهما Big endian و Little indian لتخزين كلمة من أربع بايتات في حاسوب ذي كلمة 32 بت وتعنون البايت ، وفي الطريقتين كلتيهما يكون عنوان الكلمة هو X أي عنوان البايت الأول وجرت العادة على تقدير حجم الذاكرة بالبايت Byte=8bit ، فعلى سبيل المثال ذاكرة بحجم 256Mbyte أي تحوي $256\times2^{20}=2^{28}$ byte . كما ذكرنا سابقاً تحفظ تعليمات البرنامج والمعطيات في الذاكرة ، ومن أجل القيام بعملية تخزين كلمة (كتابة) أو استرجاعها (قراءة) يرتبط بكل كلمة عنوان وحيد يستخدم لتحديد موقعها في الذاكرة ، وتستخدم لذلك الأرقام من 0 إلى M-1 وتؤلف هذه العناوين

فضاء عنوان Address Space الحاسوب ويلزم عند استخدام الترميز الثنائي للعناوين m bit حيث $M=2^m$. يجب أن يكون الحاسوب قادراً على معالجة الحروف بالإضافة إلى الأرقام ويمكن أن تكون الحروف أحرفاً أبجدية أو خانات عشرية أو علامات ترقيم كالنقطة والفاصلة وغيرها . يبين الشكل 4 أنواع محتملة للمعطيات التي يمكن أن تخزن في كلمة ذاكرة طولها 32 بت.

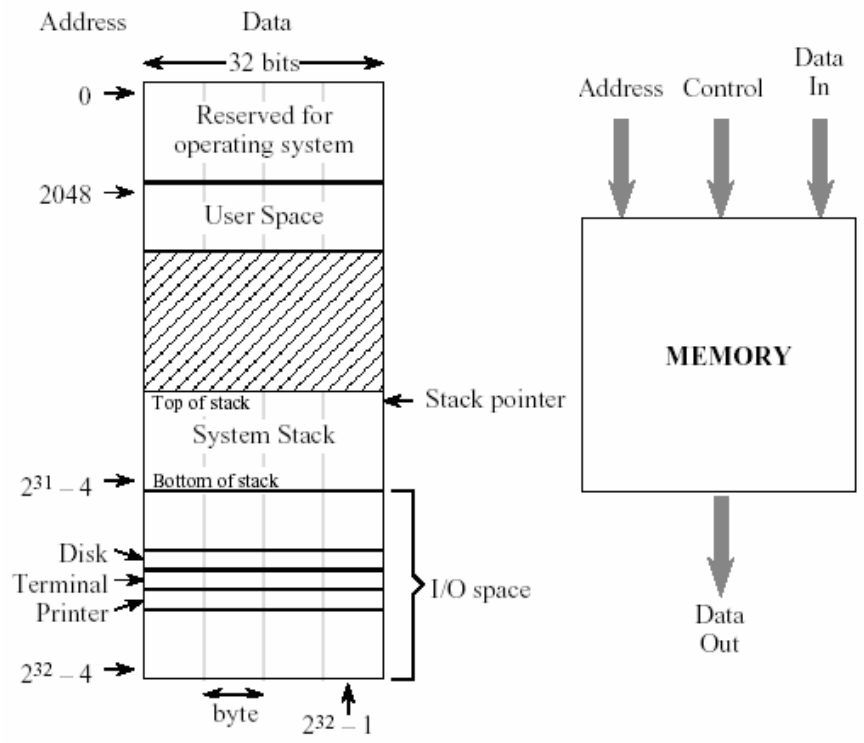
Bit	0
Nibble	0110
Byte	10110000
16-bit word (halfword)	11001001 01000110
32-bit word	10110100 00110101 10011001 01011000
64-bit word (double)	01011000 01010101 10110000 11110011 11001110 11101110 01111000 00110101
128-bit word (quad)	01011000 01010101 10110000 11110011 11001110 11101110 01111000 00110101 00001011 10100110 11110010 11100110 10100100 01000100 10100101 01010001

الشكل 1 الحجوم الشائعة للمعطيات المخزنة في موقع ذاكرة

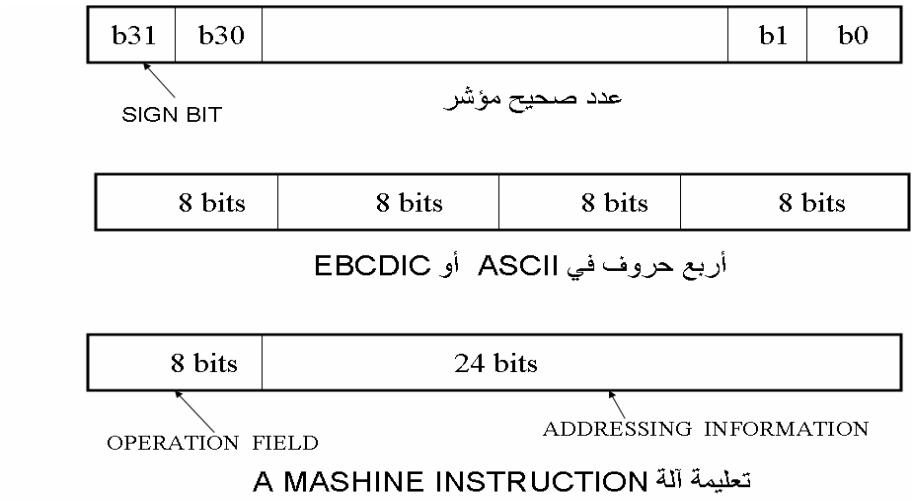


الشكل 2- أساليب تخزين كلمة ذاكرة متعددة البايتات (Big endian و Little endian)

يبين الشكل 3 مخطط افتراضي لذاكرة رئيسية حجمها $2^{32}\times32bit$ لحاسوب يعنون البايتـات (في هذه الحالة يبلغ فضاء الذاكرة 4G ويجب أن نتذكر أن مايحدد حجم فضاء الذاكرة لـيس طول الكلمة وإنما طول العنوان الذي يمكن للمعالج أن يولده ويحمله على خطوط العناوين) . يتضمن مثال الذاكرة في الشكل 3 أجزاء محجوزة لنظام التشغيل وبرامج المستخدم والمكدس وعناوين أجهزة دخل /خرج .



الشكل 3- خريطة محتملة للذاكرة الرئيسية .



الشكل 4- أنواع محتملة للمعطيات المخزنة في الذاكرة الرئيسية

3- مجموعة تعليما ت الحاسوب وصيغها :

يمكن تقسيم مجموعة تعليمات أي حاسوب حسب وظائفها إلى أربعة أنواع رئيسية هي :

1. تعليمات معالجة المعطيات : تحول هذه التعليمات المعطيات من صيغة لأخرى عبر استخدام ALU داخل المعالج ، وتشمل التعليمات الحسابية والتعليمات المنطقية وتعليمات الإزاحة والمقارنة وتعليمات الأغراض الخاصة
2. تعليمات نقل المعطيات : تنقل المعطيات من مكان إلى آخر في الحاسوب دون تغييرها ومن هذه التعليمات تعليمات تناقل معطيات الذاكرة وتعليمات الإدخال والإخراج وتعليمات النقل الداخلي وتعليمات المكدس .
3. تعليمات معالجة البرنامج : تنقل هذه التعليمات تحكم البرنامج من مكان لآخر في الذاكرة . إنها تغير عداد البرنامج بحيث تنفذ التعليمات بغير ترتيبها المعتاد وهي : تعليمات القفز الشرطي وغير الشرطي وتعليمات استدعاء الروتينات الجزئية وتعليمات التوقف .
4. تعليمات إدارة الحالة : تغير هذه التعليمات شروط حالة الحاسوب دون تغيير المعطيات أو ترتيب تنفيذ التعليمات أي تقوم بوظائف الإدارة وليس المعالجة .

يجب عنونة المعامل Operand (أو المعاملات) التي تتضمنها أي عملية يقوم بها الحاسوب ،ولهذا يجب أن تتضمن أي تعليمة يصدرها المعالج على الأقل نـوعين مـن المعلومـات : معلومات عن العملية Operation التي ستنجز ، وترمـز فـي حقـل يـسمى Opcode ، ومعلومات عن المعامل الذي ستتم عليه العملية وترمز في حقل يسمى حقل العنوان .

تصنف تعليمات الحاسوب من حيث عدد العناوين إلى :ثلاثية أو ثنائية أو أحادية أو صـفرية العنوان إضافة لتعليمات ذات عنوان ونصف وهذا يتعلق بالتنظيم الداخلي لوحـدة المعالجـة المركزية . نناقش هذه الصيغ من خلال تعليمة الجمع Add كمثال مع ملاحظة أن صـيغة التعليمة في هذه الأمثلة ستتكون من الحقول التالية :

Opcode source distenation

حيث يمثل حقل opcode ترميز العملية التـي تنـص عليهـا التعليمـة مثـل Add ، Subtract ، Write، AND ،ويمثل حقل source (المصدر) المعامـل (أو المعاملات) وهو قد يكون عدد ثابت أو قيمة مخزنة في سجل او في الـذاكرة ، وأخيـراً يمثل حقل Destination (الوجهة) المكان الذي ستحفظ فيه نتيجة العملية .

إن أبسط صيغة للجمع هي [A]+[B]→[C] التي تعني جمع محتوى موقع الـذاكرة ذي العنوان A إلى محتوى موقع الذاكرة ذي العنوان B ووضع الناتج في موقع الذاكرة ذي

العنوان C . وللتعبير عن هذه العملية بتعليمة واحدة يلزمنا حقل عنوان يتضمن عناوين المعاملات الثلاثة بالإضافة لحقل ترميز العملية Opcode وتكون صيغة التعليمة بالشكل : Add A,B,C حيث يسمى المعاملان A،B معاملي المصدر والمعامل C معامل الوجهة . إن سيئة هذه التعليمة هي طول حقل العنوان لذلك تستخدم بعض الحواسيب تنظيماً للمعالج يسمح بتنفيذ عملية الجمع هذه بتعليمات ثنائية العنوان وذلك بأن تعيد ناتج الجمع إلى أحد موقعي المعاملين مدمرة بذلك أحد المعاملين الأصليين وتكون صيغة التعليمة هي Add A،B والتي تقوم بالعملية [A]+[B]→[B] ،ولكن لايمكن استخدام هذه التعليمة وحدها لحل المسألة السابقة بل نحتاج تعليمة أخرى ثنائية العنوان تقوم بنسخ محتويات موقع من الذاكرة إلى موقع آخر وهي تعليمة النقل Mov B,C والتي تقوم بالعملية [B]→[C] . الاحتمال التالي هو استخدام تعليمات أحادية العنوان ، ولما كانت عملية الجمع تتطلب معاملين يتوجب إعادة تنظيم المعالج بحيث يكون موقع المعامل الآخر معروفاً وكذلك مكان حفظ النتيجة . يستخدم لهذا الغرض أحد سجلا ت الأغراض العامة ويسمى المراكم Accumulator كمكان لحفظ المعامل الثاني وكذلك مكان لحفظ النتيجة وتعني عندئذ التعليمة Add A جمع محتوى الموقع A إلى محتوى المراكم ووضع الناتج في المراكم . وعندئذٍ لتنفيذ عملية الجمع [A]+[B]→[C] يلزمنا ثلاث تعليمات أحادية العنوان حسب التسلسل التالي :

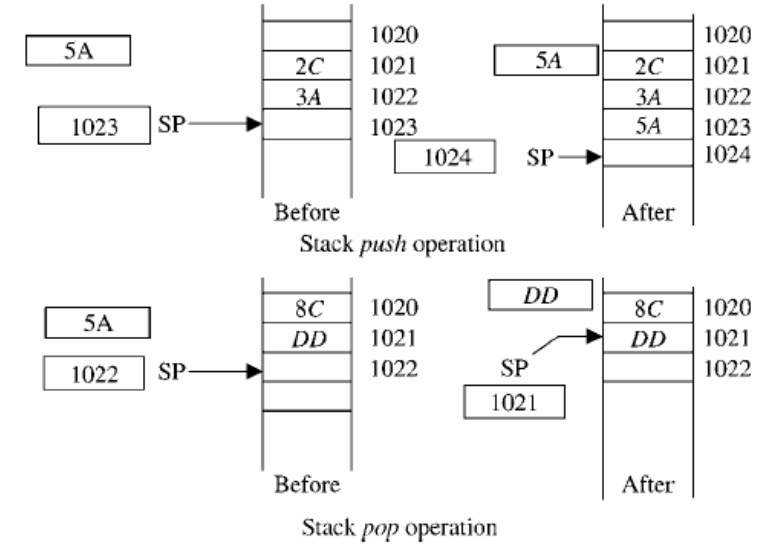
Load A
Add B
Store C

حيث تقوم التعليمة Load A بنقل محتويات الموقع A إلى المراكم وتقوم التعليمة Store C بحفظ محتويات المراكم في الموقع C من الذاكرة . وهنا .يجب الانتباه إلى أن المعامل المحدد في حقل المعامل يمكن أن يكون وجهة أو مصدر اً ففي التعليمة Load A يحدد العنوان A مصدر المعامل بينما يكون المراكم هو الوجهة ضمناً ، أما في لتعليمة Store C فالعنوان C هو الوجهة والمراكم هو المصدر ضمناً .

تملك معظم الحواسيب عدداً من سجلات الأغراض العامة في وحدة المعالجة المركزية يمكن استخدامها بالطريقة ذاتها التي يستخدم فيها المراكم وسيلزم 3 بتات (إذا كان عدد السجلا ت8) أو 4 بتات (إذا كان عدد السجلا ت 16) في حقل ما من التعليمة لعنونة السجل الذي سيشترك بالعملية . نلاحظ أن عدد هذه البتات أقل بكثير من عدد البتات اللازمة لعنوان موقع في الذاكرة وهنا سنحصل على صيغة تعليمة ذات العنوان والنصف مثال هذه التعليمات :

Load A, Ri ، Store R, B , Add Ri,A

حيث Ri أحد سجلات الأغراض العامة .كما تتضمن الحواسيب تعليمات تسمح بإجراء عمليات بين السجلات نفسها مثل التعليمة Add Ri,Rj التي تضيف محتويات السجل Ri إلى محتويات السجل Rj وتضع الناتج في Rj .وأخيرا يمكن استخدام تعليمات تعرف فيها مواقع جميع المعاملات ضمناً حيث تعتمد هذه التعليمات على استخدام مكدس ويشار عادة إلى مثل هذه التعليمات بتعليمات صفرية العنوان . فالمكدس هو آلية لتنظيم المعطيات بحيث أن المعطيات المخزنة أخيراً هي التي تستعاد أولاً (LIFO) وهناك عمليتان أساسيتان على المكدس :PUSH وهي الحفظ (الكتابة) في المكدس و POP وهي القراءة من المكدس ويبين الشكل 5 هاتين العمليتين . واضح من الشكل 5 أن هناك سجلاً خاصاً يدعى مؤشر المكدس SP يستخدم ليشير إلى عنوان الموقع (قمة المكدس) الذي ستحفظ فيه القيمة 5A وحسب الشكل هذا العنوان هو 1023 . بعد إتمام عملية الحفظ تزداد محتويات المكدس بمقدار 1 لتشير إلى العنوان الجديد لقمة المكدس (1024) . أما في عملية القراءة POP فيجري إنقاص SP أولاً وتخزن القيمة المطلوبة في الموقع الجديد ((حسب الشكل تخزن القيمة DD في الموقع 1022)) .



الشكل 5- تعليمتا الكتابة Push والقراءة Pop في المكدس

4- أنماط العنونة :

يقصد بأنماط العنونة الطرق المتبعة لتحديد عناوين موقع الذاكرة أو السجلات التي تستخدمها التعليمة ويجري الاختيار بين هذه الطرق حسب الخصائص المرغوب تحقيقها في طريقة العنونة . ويطلب أحياناً أن يكون العنوان أقصر ما يمكن أو يكون هناك إمكانية لتغيير العنوان الفعلي دون تغيير التعليمة بحيث يمكن استخدام تتابع واحد من التعليمات لمعالجة جميع عناصر مصفوفة أو جدول أو قائمة . وعموماً تفضل الطرق التي تنفذ بسرعة والطرق البسيطة على الطرق المعقدة . يبين الشكل 7 تمثيلاً لأهم أنماط العنونة و فيما يلي نوضح معنى هذه الأنماط .

- **العنونة المباشرة (المطلقة) Direct (Absolute) Mode**

يعطى في هذا النمط عنوان موقع المعامل (العنوان الفعلي) صراحة كجزء من التعليمة . جميع التعليمات التي وردت في الفقرة السابقة هي ذات عنونة مباشرة . يتميز هذا النمط بالبساطة إذ أنه لا يتطلب أي حسابات إضافية ولكن يفتقد هذا النمط المرونة اللازمة لمعالجة الجداول والقوائم وقد يكون العنوان طويلاً .

- **نمط العنونة غير المباشر :**

هنا العنوان الفعلي للمعامل هو محتويات سجل من سجلات الأغراض العامة أو محتويات موقع من الذاكرة يظهر عنوانه كجزء من التعليمة .مثال:

Add (A) , R0 أو Add (R0), A

- **النمط الفوري Immidiate addressing mode**

يعطى في هذا النمط المعامل صراحة كجزء من التعليمة ويستخدم الرمز # للدلالة على هذا النمط فالتعليمة Add # 500 , Ri تقوم بإضافة القيمة 500 إلى محتوى السجل Ri وتضع الناتج في Ri .

- **نمط الفهرسة Indexed mode**

يعني هذا النمط أن وحدة المعالجة المركزية تولد العنوان الفعلي للمعامل بإضافة محتويات سجل ما من سجلات الأغراض العامة إلى قيمة ثابتة تظهر في التعليمة وتكون صيغة هذا النمط كما يلي:

Op code Value (Ri) ، Rj

حيث Ri هو سجل الفهرسة . مثال ذلك التعليمة Load 100 (R1), R2

تقوم بإضافة القيمة 100 الى محتويات سجل الفهرسة R1 لتوليد العنوان الفعلي للمعامل ثم تحميله في السجل R2 .

- **- نمط العنونة النسبي Relative addressing mode**

تضيف ال CPU محتويات عداد البرنامج PC إلى العنوان الذي يرد في التعليمـة لإيجـاد العنوان الفعلي للمعامل . يستخدم الرمز $ للإشارة إلى القيمة الجارية لعـداد البرنامـج مثلاً ، التعليمة Load $ +100 تقوم باضافة القيمة 100 إلى محتوى عداد البرنامج ثم وضع النتيجة في المراكم(AC) . يبين الشكل 6 (أ- ب-ج-د-هـ) تمثيلاً لأنماط العنونـة السابقة .

- **نمط عنونة التزايد الآلي Autoincrement addressing mode**

العنوان الفعلي في هذا النمط هو محتويات سجل من سجلات الأغراض العامة تحدده التعليمة وتزداد محتويات هذا السجل مباشرة بعد النفاذ للمعامل لتشير إلى الفقرة التالية في قائمـة مـا يكتب هذا النمط بالشكل: +(Ri) . يبين الشكل 7 تمثيلاً للتعليمة :

Load (Rauto)+, Ri

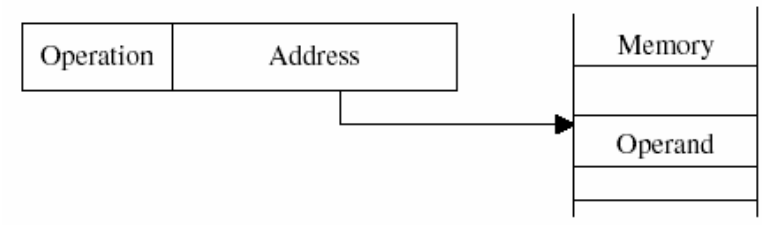
قبل وبعد التنفيذ .

- **نمط عنونة التناقص الآلي Autdecrement addressing mode**

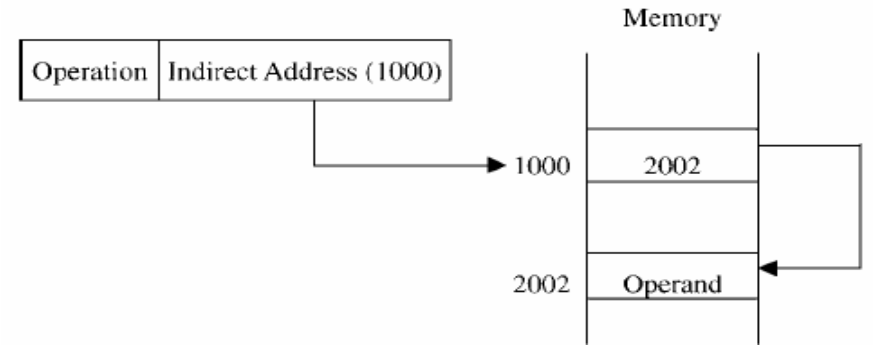
هنا يتم إنقاص محتويات السجل المذكور في التعليمة ثم تستخدم هذه المحتويات كعنوان فعلي للمعامل ويكتب بالشـكل: -(Rj) . ويبين الشكل 9 تمثيلاً للتعليمـة

Load -(Rauto), Ri

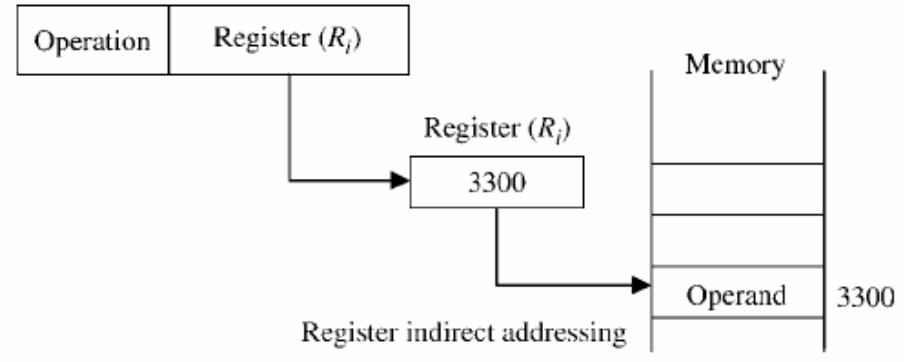
قبل وبعد التنفيذ . يتضمن الجدول 1 تعريفاً لأنماط العنونة وتوضيحها من خلال العملية التي يقوم بها ومثال على كل منها.



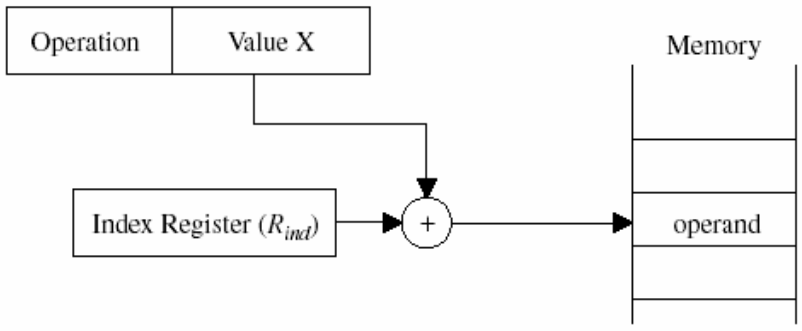
أ- نمط العنونة المباشر



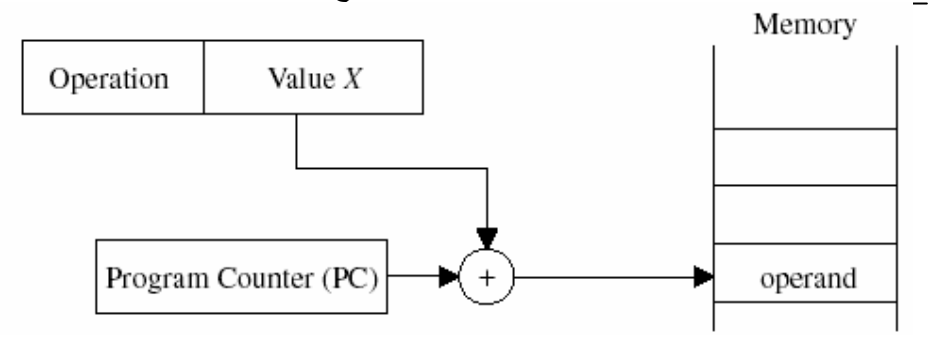
ب- نمط العنونة غير المباشر



ج- نمط عنونة سجلات غير مباشر

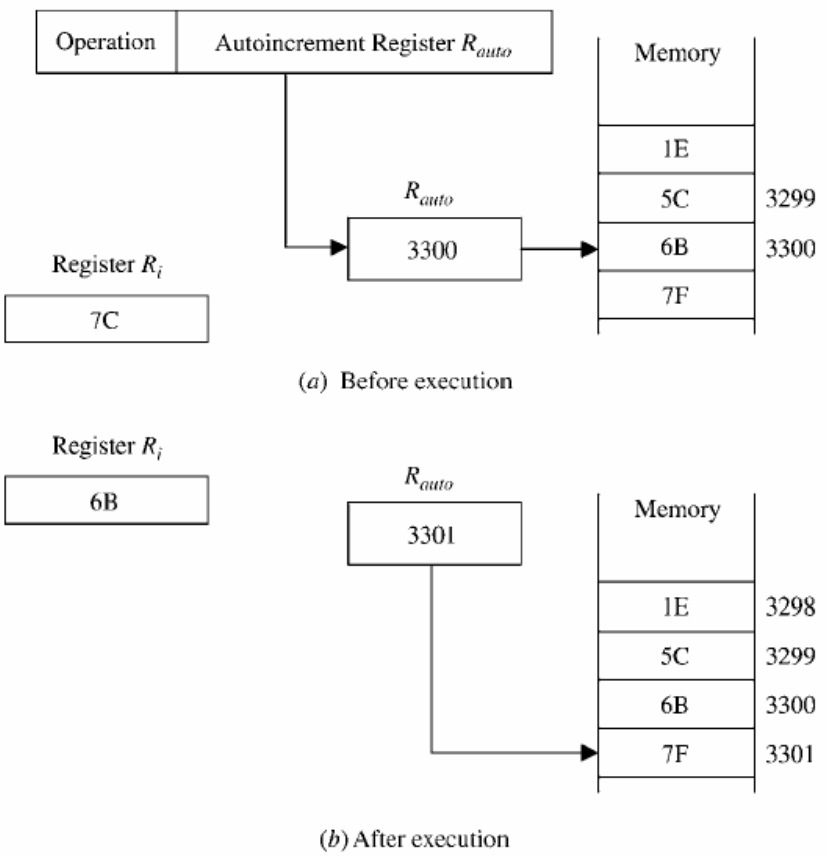


د-نمط الفهرسة

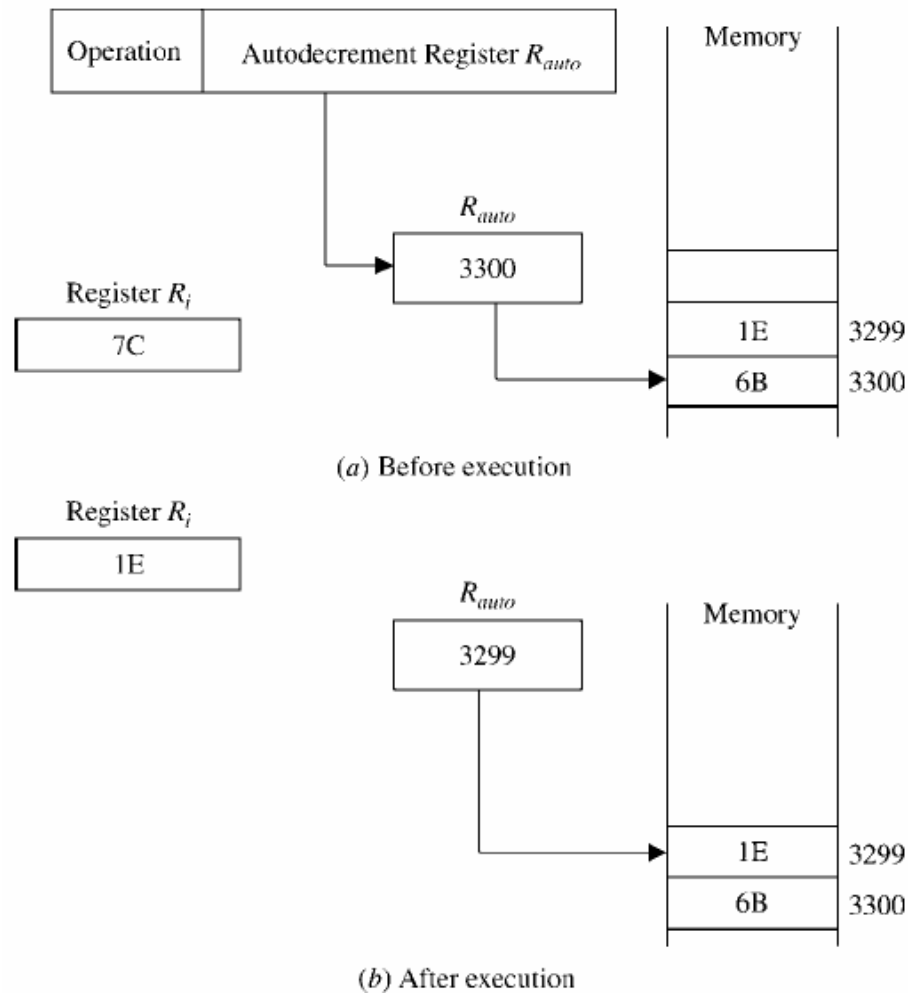


هـ - نمط العنونة النسبي

الشكل 6



الشكل 7– توضيح لنمط عنونة التزايد الآلي قبل وبعد تنفيذ التعليمة
Load (Rauto)+, Ri



الشكل 8 – توضيح لنمط عنونة التناقص الآلي قبل وبعد تنفيذ التعليمة
Load -(Rauto), Ri

5- أمثلة برمجية :

نوضح من خلال الأمثلة التالية بعض المفاهيم مثل الأعلام وتعليمات استدعاء روتينات جزئية كذلك أثر استخدام أنماط عنونة مختلفة على البرنامج .

1- يمثل العلم flag أحد بتات سجل خاص يدعى سجل حالة البرنامج وقيم هـذه البتـات (الأعلام) تتعلق بنتيجة تنفيذ تعليمات مختلفة في سياق تنفيذ البرنامج .

Addressing mode	Definition	Example	Operation
Immediate	Value of operand is included in the instruction	*load* #1000, R_i	$R_i \leftarrow 1000$
Direct (Absolute)	Address of operand is included in the instruction	*load* 1000, R_i	$R_i \leftarrow M[1000]$
Register indirect	Operand is in a memory location whose address is in the register specified in the instruction	*load* (R_j), R_i	$R_i \leftarrow M[R_j]$
Memory indirect	Operand is in a memory location whose address is in the memory location specified in the instruction	*load* (1000), R_i	$R_i \leftarrow M[1000]$
Indexed	Address of operand is the sum of an index value and the contents of an index register	*load* $X(R_{ind})$, R_i	$R_i \leftarrow M[R_{ind} + X]$
Relative	Address of operand is the sum of an index value and the contents of the program counter	*load* $X(PC)$, R_i	$R_i \leftarrow M[PC + X]$
Autoincrement	Address of operand is in a register whose value is incremented after fetching the operand	*load* $(R_{auto})+$, R_i	$R_i \leftarrow M[R_{auto}]$ $R_{auto} \leftarrow R_{auto} + 1$
Autodecrement	Address of operand is in a register whose value is decremented before fetching the operand	*load* $-(R_{auto})$, R_i	$R_{auto} \leftarrow R_{auto} - 1$ $R_i \leftarrow M[R_{auto}]$

الجدول 1 تعريف أنماط العنونة وتوضيحها من خلال العملية التي يقوم بها ومثال على كل منها.

نبين في المقطع التالي مفهوم الأعلام flags حيث هناك علم الصفر Z وعلم الطفح V وعلم المنقول C وعلم السالب N وغيرها ..يقوم المعالج باختبار هذه الأعلام عند اللـزوم ليقـرر الخطوة التالية في المعالجة حسب قيمة العلم بحيث يضمن التنفيذ الصحيح للتعليمـات .يبـين الجدول 2 توضيحاً لمعنى هذه الأعلام .

Flag name	Meaning
Negative (N)	Set to 1 if the result of the most recent operation is negative, it is 0 otherwise
Zero (Z)	Set to 1 if the result of the most recent operation is 0, it is 0 otherwise
Overflow (V)	Set to 1 if the result of the most recent operation causes an overflow, it is 0 otherwise
Carry (C)	Set to 1 if the most recent operation results in a carry, it is 0 otherwise

الجدول 2 توضيح معنى مجموعة من الأعلام الشهيرة

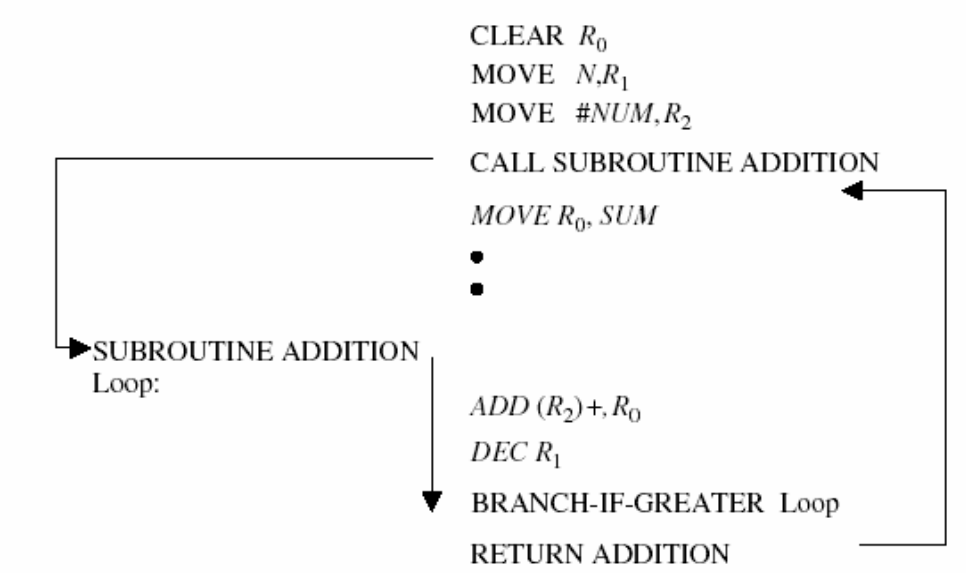
فيما يلي مقطع من برنامج يجري فيه اختبار علم الصفر .

```
        LOAD                #100, R1
Loop:   ADD                 (R2)+, R0
        DECREMENT           R1
        BRANCH-IF-GREATER-THAN Loop
```

يتبين من هذا المقطع أن التعليمة الرابعة هي تعليمة تفريع شرطي والتي تشير إلى أنه طالما كان محتوى السجل R1 الذي يجري إنقاصه في التعليمة السابقة أكبر من الـصفر يجـري التفريع إلى التعليمة ذات العنوان Loop أي أن المعالج يختبر العلم Z قبل أن يقرر الانتقـال إلى هدف التفريع (Z=0)أو التعليمة التالية لتعليمة التفريـع الشرطي (Z=1) ، وهـذا هـو أسلوب تنفيذ التعليمات الشرطية .

2- إن لتعليمة استدعاء روتين جزئي Call نفس أثر تعليمة القفز (التفريع) اللاشرطي مـن حيث تحميل عداد البرنامج بعنوان هدف التفريع .ومن أجل العودة يخـزن العنـوان التـالي لتعليمة Call(عنوان العودة) في المكدس لكي تقوم تعليمة Return في نهاية الروتين الجزئي بعملية POP على المكدس يتم بنتيجتها إعادة تحميل عداد البرنامج بعنوان العودة .يبين المثال رقم 2 مقطعاً من برنامج يستخدم تعليمة Call لروتين جزئي يسمى ADDITION يقـوم بجمع عدد N من القيم موجودة في مواقع متسلسلة من الذاكرة تبدأ بـالموقع NUM .يقـوم الروتين الجزئي ADDITION هنا بعملية الجمع بينما يحفظ البرنامج الأب النتيجـة فـي موقع الذاكرة ذي العنوان SUM ، وتؤمن التعليمـة RETURN ADDITION العـودة للبرنامج الأب .

المثال رقم 2



3- نوضح في المثال 3 بعض أنماط العنونة وأثرها على حجم البرنامج من خلال كتابة برنامجين لهما الوظيفة نفسها :جمع 100عدد مخزنة في الذاكرة ابتداء من الموقع 1000وحفظها في الموقع 2000 .

المثال 3:

	$CLEAR\ R_0;$	$R_0 \leftarrow 0$
	$MOVE\ \#100, R_1;$	$R_1 \leftarrow 100$
	$CLEAR\ R_2;$	$R_2 \leftarrow 0$
$LOOP:$	$ADD\ 1000(R_2), R_0;$	$R_0 \leftarrow R_0 + M\ (1000 + R_2)$
	$INCREMENT\ R_2;$	$R_2 \leftarrow R_2 + 1$
	$DECREMENT\ R_1;$	$R_1 \leftarrow R_1 - 1$
	$BRANCH\text{-}IF > 0\ LOOP;$	$GO\ TO\ LOOP\ if\ contents\ of\ R_1 > 0$
	$STORE\ R_0, 2000;$	$M(2000) \leftarrow R_0$

استخدم في هذا البرنامج نمط العنونة الفوري من خلال التعليمة:

MOVE#100،R1

ونمط الفهرسة من خلال التعليمة:

Add 1000R(2)،R0.

يمكن كتابة البرنامج نفسه باستخدام نمط التزايد الآلي ونحصل على عدد تعليمات أقل كما يلي:

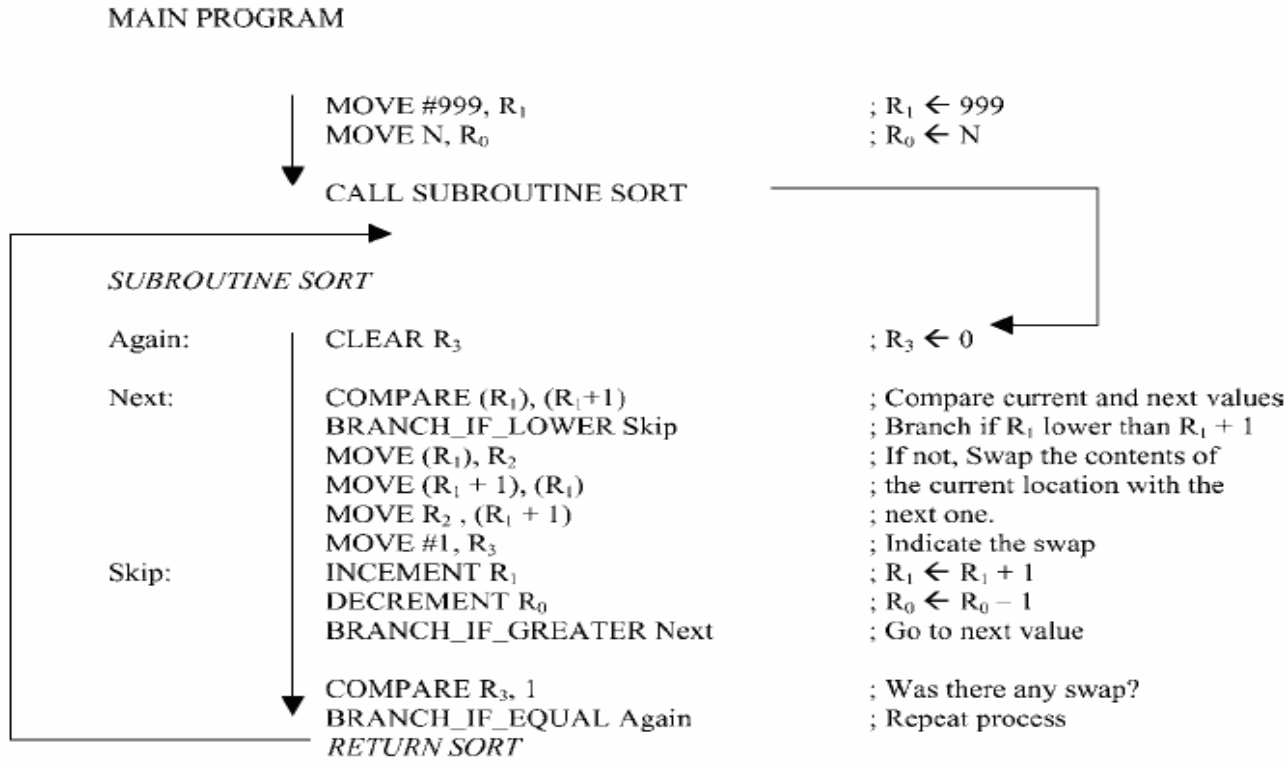
	CLEAR R_0;	$R_0 \leftarrow 0$
	MOVE #100, R_1;	$R_1 \leftarrow 100$
	CLEAR R_2;	$R_2 \leftarrow 0$
LOOP:	*ADD* $1000(R_2)+, R_0$;	$R_0 \leftarrow R_0 + M\ (1000 + R_2)\ \&\ R_2 \leftarrow R_2 + 1$
	DECREMENT R_1;	$R_1 \leftarrow R_1 - 1$
	BRANCH-IF > 0 *LOOP*;	*GO TO LOOP if contents of* $R_1 > 0$
	STORE R_0, *2000*;	$M(2000) \leftarrow R_0$

4- يوضح البرنامج التالي (المثال 4) ترتيب قائمة من N قيمة مخزنة ابتداء من الموقع 1000ترتيباً تصاعدياً وحفظها أيضاً ابتداء من الموقع 1000 باستخدام الروتين الجزئي SORT .يقوم الروتين الجزئي SORT بعمله اعتماداً على تقنية معروفة تسمى Bubble Sort فقاعة الترتيب. يجري اختبار محتوى R3 مع نهاية كل حلقة لاكتشاف حدوث الترتيب أم لا .

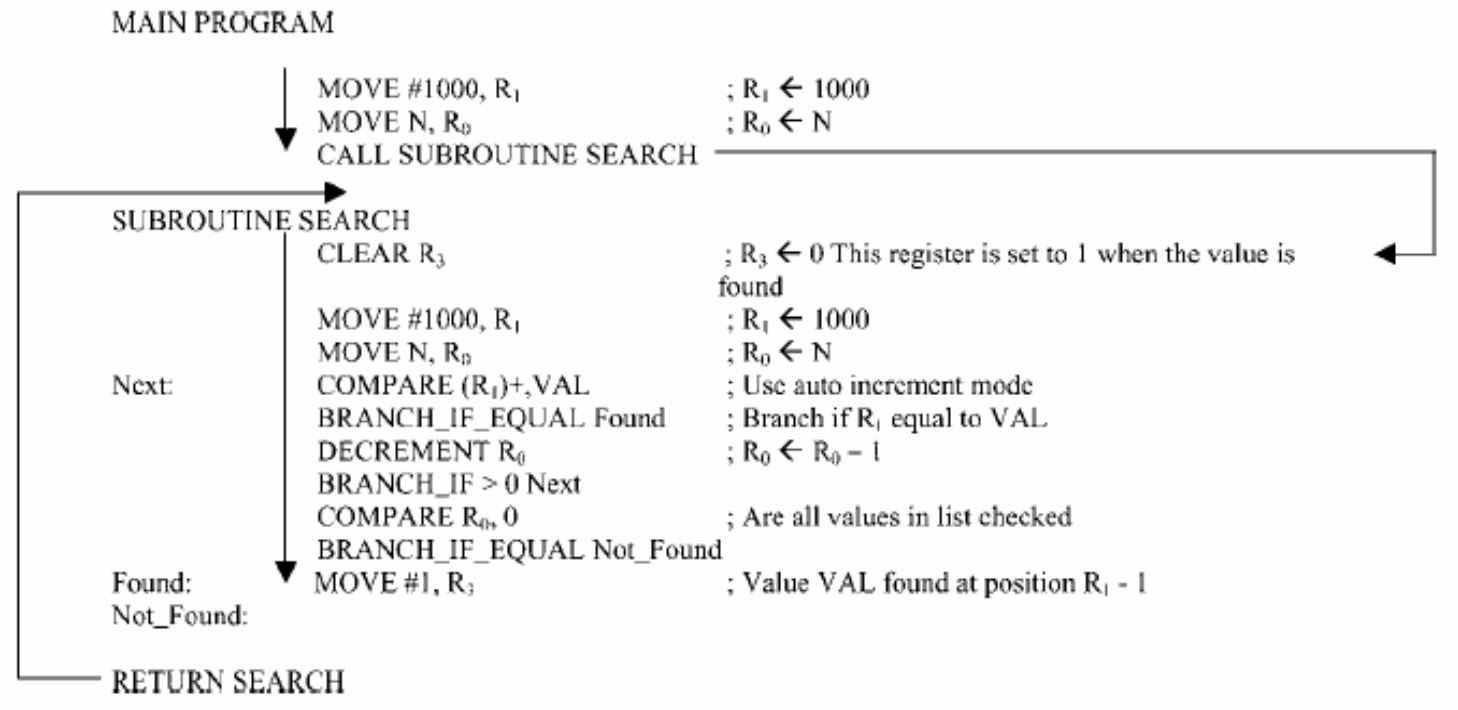
5- يستخدم المثال 5 الروتين SEARCH للبحث عن قيمة VAL من قائمة .سنفترض أن القائمة غير مرتبة .هنا تجري مقارنة القيمة VAL مع كل عنصر في القائمة ابتداء من قمة القائمة ويشير محتوى السجل R3 فيما إذا تم إيجاد القيمة VAL أم لا .إن أول عنصر في القائمة مخزن في الموقع 1000 .

6- يبين المثال 6 استخدام المكدس لتنفيذ الروتين SEARCH حيث يرسل البارمترين N،VAL إلى المكدس .

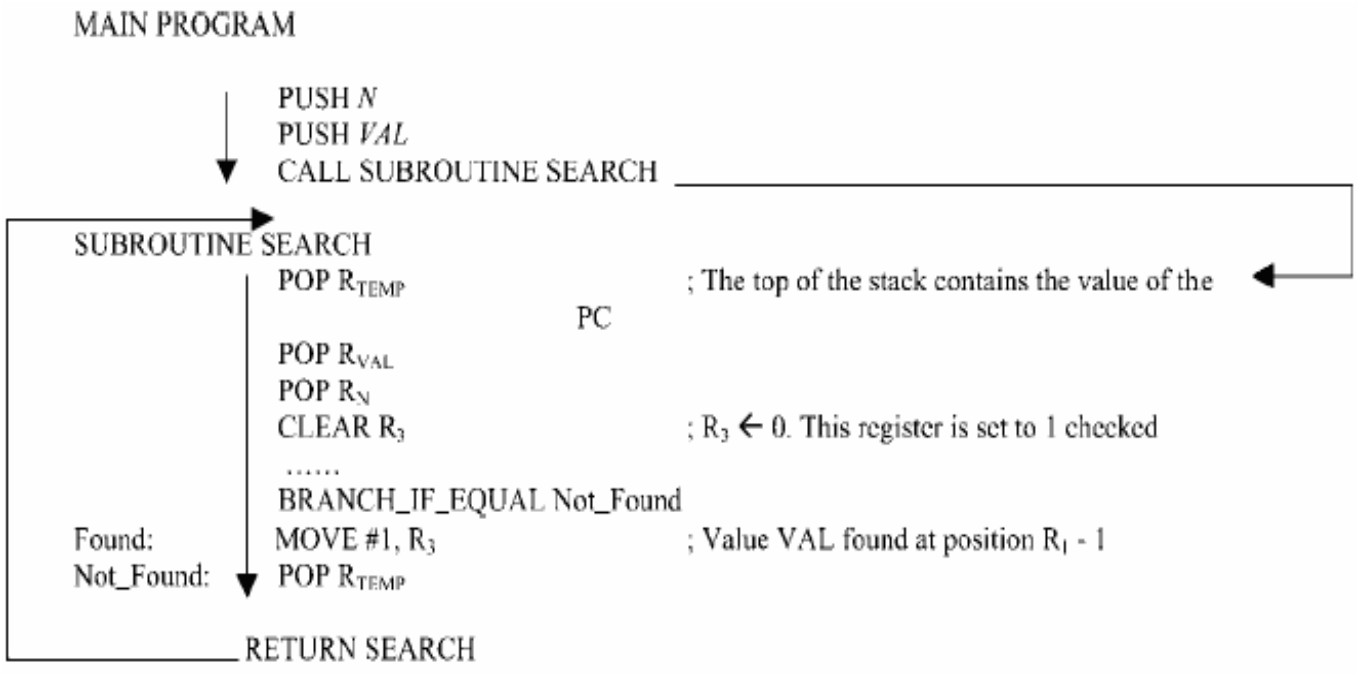
المثال 4


MAIN PROGRAM
MOVE #999, R1 ; R1 ← 999
MOVE N, R0 ; R0 ← N
CALL SUBROUTINE SORT
SUBROUTINE SORT
Again: CLEAR R3 ; R3 ← 0
Next: COMPARE (R1), (R1+1) ; Compare current and next values
BRANCH_IF_LOWER Skip ; Branch if R1 lower than R1 + 1
MOVE (R1), R2 ; If not, Swap the contents of
MOVE (R1 + 1), (R1) ; the current location with the
MOVE R2 , (R1 + 1) ; next one.
MOVE #1, R3 ; Indicate the swap
Skip: INCEMENT R1 ; R1 ← R1 + 1
DECREMENT R0 ; R0 ← R0 – 1
BRANCH_IF_GREATER Next ; Go to next value
COMPARE R3, 1 ; Was there any swap?
BRANCH_IF_EQUAL Again ; Repeat process
RETURN SORT

المثال 5

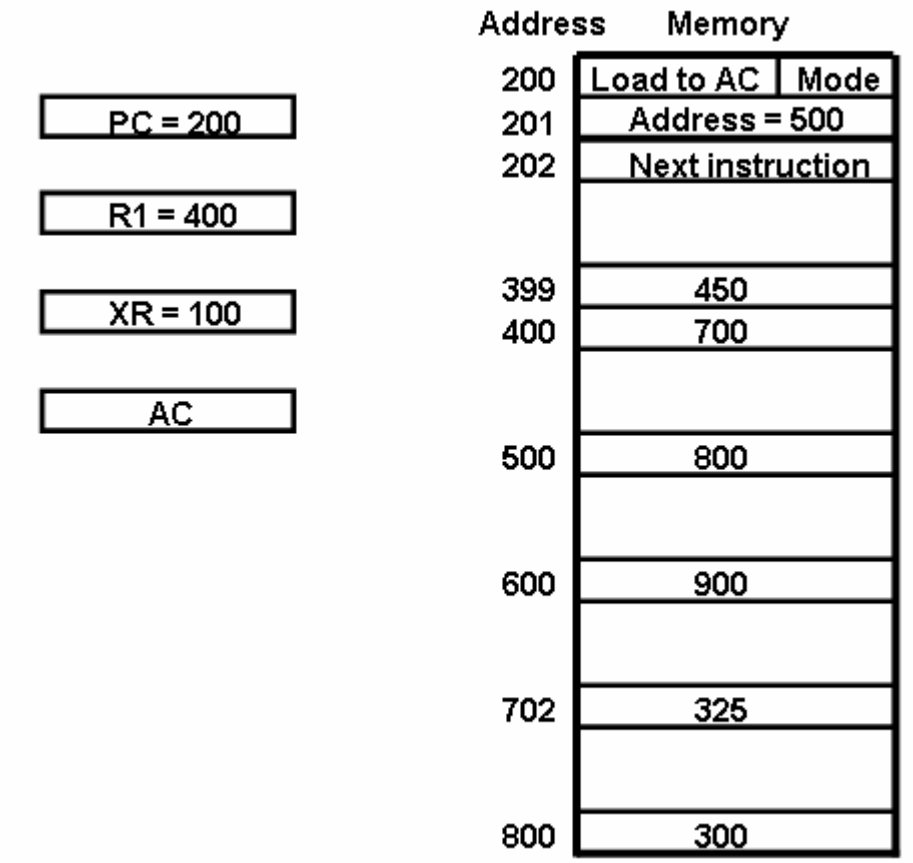

MAIN PROGRAM
MOVE #1000, R1 ; R1 ← 1000
MOVE N, R0 ; R0 ← N
CALL SUBROUTINE SEARCH
SUBROUTINE SEARCH
CLEAR R3 ; R3 ← 0 This register is set to 1 when the value is found
MOVE #1000, R1 ; R1 ← 1000
MOVE N, R0 ; R0 ← N
Next: COMPARE (R1)+,VAL ; Use auto increment mode
BRANCH_IF_EQUAL Found ; Branch if R1 equal to VAL
DECREMENT R0 ; R0 ← R0 – 1
BRANCH_IF > 0 Next
COMPARE R0, 0 ; Are all values in list checked
BRANCH_IF_EQUAL Not_Found
Found: MOVE #1, R3 ; Value VAL found at position R1 - 1
Not_Found:
RETURN SEARCH

المثال 6 :


MAIN PROGRAM
PUSH N
PUSH VAL
CALL SUBROUTINE SEARCH
SUBROUTINE SEARCH
POP R_TEMP ; The top of the stack contains the value of the PC
POP R_VAL
POP R_N
CLEAR R_3 ; R_3 ← 0. This register is set to 1 checked
.....
BRANCH_IF_EQUAL Not_Found
Found: MOVE #1, R_3 ; Value VAL found at position R_1 - 1
Not_Found: POP R_TEMP
RETURN SEARCH

6- أسئلة وتمرينات :

1 –احسب العنوان الفعلي ومحتوى المراكم AC ونظم النتائج ا في الجدول المبين على يسار الشكل وذلك عند تنفيذ التعليمة Load (LD) ذات الصيغة المبينة في جزء الذاكرة الموضـح في الشكل.اعتبر XR سجل فهرسة .



Addressing Mode	Effective Address	Content of AC
Direct address		
Immediate operand		
Indirect address		
Relative address		
Indexed address		
Register		
Register indirect		
Autoincrement		
Autodecrement		

2- ليكن التعبير الحسابي :**X = (A + B) * (C + D)** والمطلوب :

أولاً : اكتب باستخدام تعليمات اللغة التجميعية Assembly التي تعلمتها حتى الآن برنامجاً لحساب **هذا التعبير وذلك في الحالات التالية :**

- باستخدام تعليمات ثلاثية العنوان
- باستخدام تعليمات ثنائية العنوان
- باستخدام تعليمات أحادية العنوان
- باستخدام تعليمات صفرية العنوان

علماً أن A,B,C,D,X تمثل عناوين مواقع في الذاكرة .

ثانياً :نظم جدولاً تبين فيه عدد مرات النفاذ إلى الذاكرة وعدد التعليمات في كل حالة من الحالات الأربعة في الطلب الأول

ثالثاً : دون استنتاجاتك من خلال مقارنة الإحصائيات التي حصلت عليها في الطلب الثاني .

3- ما تأثير استخدام أنماط عنونة مختلفة على البرنامج . وضح ذلك بمقارنة بين الأمثلة الواردة في هذا الفصل .

4- لماذا تستخدم الأعلام في البرنامج ؟ اذكر أربعة أعلام شهيرة ومعنى كل منها .

5- ما المقصود بمجموعة تعليمات الحاسوب ؟ هل يمكن لحاسوب لا يملك في مجموعة تعليماته تعليمة ضرب MUL أن يضرب عددين .؟

6- ما العلاقة بين فضاء ذاكرة الحاسوب وعرض خطوط العنونة .

7- ماهو عنوان البايت الأول والأخير من الكلمة ذات العنوان 256 في حاسوب يعنون البايت وله طول كلمة32 بت ؟ ماهو فضاء الذاكرة ؟ وما حجم الذاكرة بالبايت ؟