4

17

2040 sp

نظري

كلية الهندسة المعلوماتية

السنة الثالثة

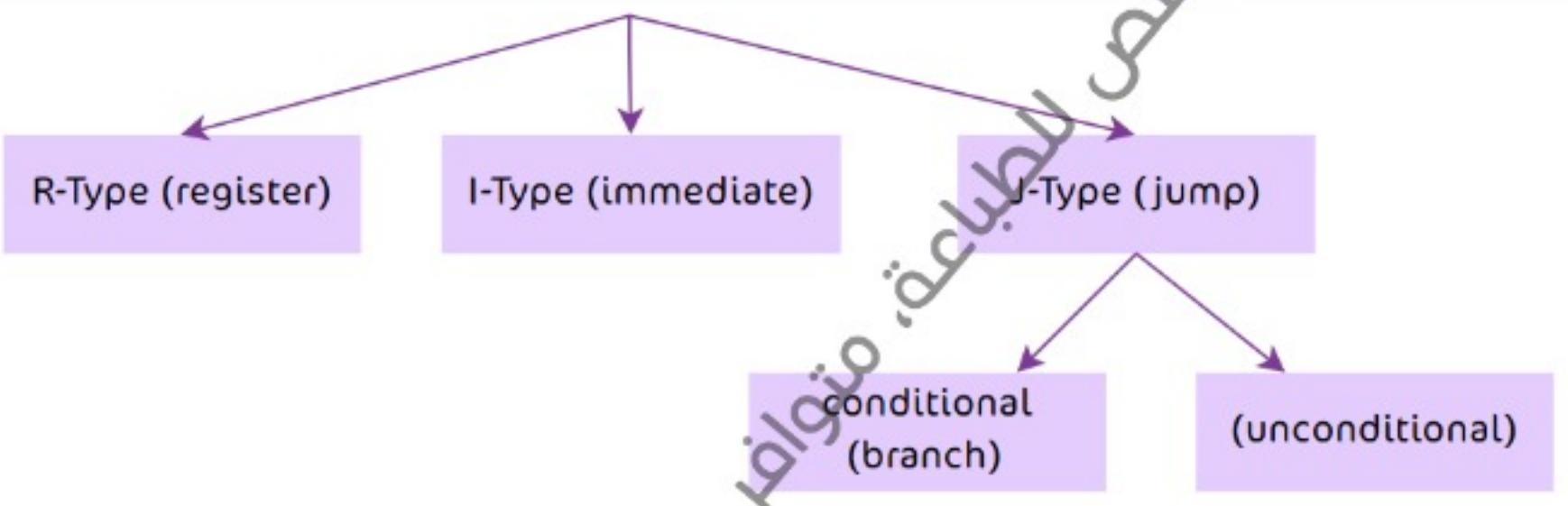
Control Unit Design

د. سيرا أستور

RB Informatics; 21/10/2024

بنيان الحواسيب 2

سندرس تصميم الـ control unit لمعالجات الـ MIPS، نتذكر أنه يوجد ثلاثة أنواع من التعليمات:



عند تصميم معالج ما بغض النظر عن نوعه، فيوجد قسمين أساسيين يجب الانتباه لهما: Data path, Control

Data path and control

Data path: هو الطريق الذي تسلكه البيانات وهذا الطريق يمر بالعديد من قطع الحاسوب على سبيل المثال: الذاكرة – السجلات – ALU – قنوات الاتصال.

حيث أنه كل خطوة في معالجة عملية ما (مثل: جلب العملية – فك التشفير – التنفيذ – حفظ النتائج) تتطلب نقل بيانات (أي اتصال) و تتطلب ممرات بين الذواكر و السجلات و وحدة الحساب و المنطق.

Control: إن ممر التعليمة هو نظام تم فرضه من قبل إشارات التحكم (كل تعليمة لها ممر خاص) حيث تنظم هذه الإشارات اتجاهات تدفق البيانات في قنوات الاتصال و تختار الوظائف المناسبة لكل من الذاكرة و وحدة الحساب و المنطق.

- تولد إشارات التحكم بواسطة وحدة التحكم، و تتألف من واحدة أو أكثر من الآلات منتهية الحالة (Finite State Machines (FSM)).

Design Process

- التصميم هو عملية مبتكرة وليست بسيطة، كما نعلم أن التصميم له أكثر من مستوى تجريد (فهم التركيب).
- يوجد طريقتين للتصميم:

1. Top Down:

تتم دراسة التصميم من الوظائف المعقدة و تحليلها إلى وظائف أولية بسيطة.

مثلاً كتصميم cpu نبدأ من أن cpu تنقسم لـ control و Data path و كل واحدة منهما ما وظائفها و إلى ماذا تنقسم و هكذا.

2. Bottom Up:

تتم دراسة التصميم من بنيته الأولية إلى أن نصل للأجزاء المعقدة منه.

مثلاً هنا لدينا اجتماع عدة بوابات منطقية تعطي ALU و السجلات مكونة من قلابات flipflops (نتذكر أن القلابات هي وحدات تخزين حيث أن كل قلاب يخزن بت واحد) و هكذا... و اجتماع كل من ALU, Regs, Shifter يعطي الـ Data path، و هكذا نحو الأعلى بالتصميم إلى أن نصل لوحدة المعالجة المركزية.

تذكرة أبسيزية:

نتذكر أنه لدينا:

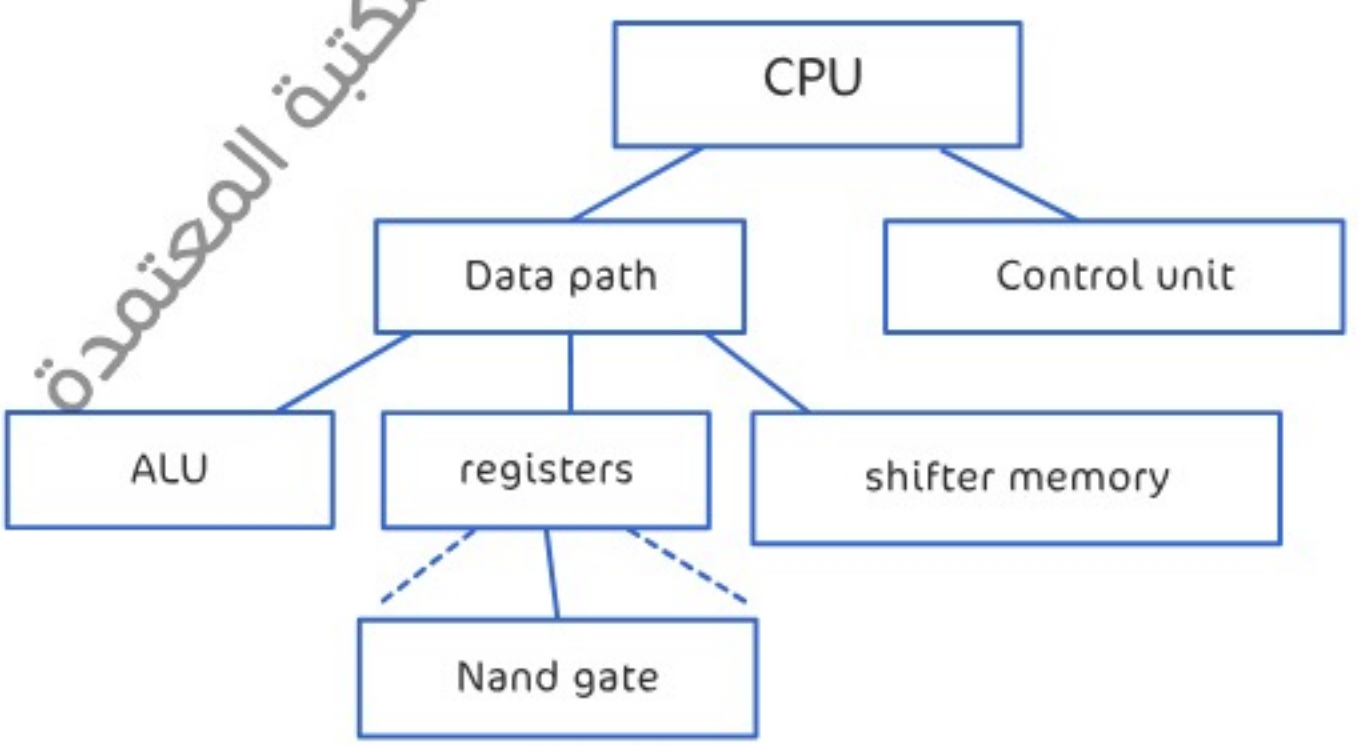
منطق تركيبي combinational: مثل بوابات and, or, Nand (لا يخزن بيانات)

منطق تتابعي Sequential: مثل القلابات flipflops (يخزن بيانات)

ماذا يعني منطق تركيبي وتتابعي؟

تركيبي: أي أن الخرج يتعلق بدخل الحالة الحالية فقط.

تتابعي: الخرج يتعلق بدخل الحالة الحالية وأيضًا خرج الحالة السابقة.



(بوابة منطقية)

Defining Control

- الأنظمة التتابعية تضم حالة مخزنة في عناصر الذاكرة الداخلية في النظام (عناصر حالة).
- سلوك هذه الدارات التتابعية يعتمد على الدخل الحالي ومضمون الذاكرة الداخلية (المخزن من الحالة السابقة) أو ما يسمى بحالة النظام.
- الدارة التتابعية لا تمثل بجدول حقيقة، و إنما تمثل بـ Finite State Machine (FSM) أي أن عدد الحالات الممكنة محدود.

الأجهزة محدودة الحالة Finite State Machine (FSM)

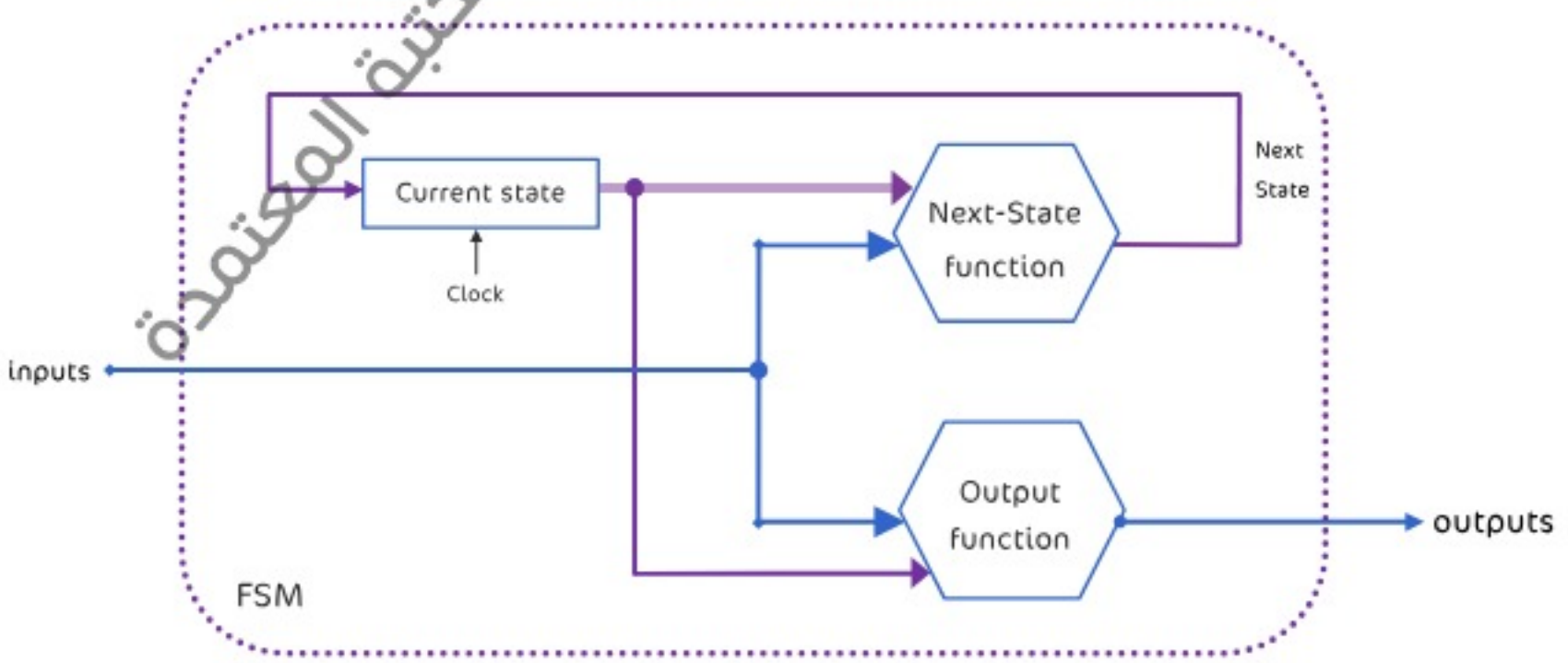
- تتألف من مجموعة حالات واتجاهات تبين تغير الحالة.
- يتم تعريف الاتجاهات باستخدام توابع الحالة التالية و الخرج.
- كما نلاحظ أننا نحتاج لمعلومات مخزنة في الذاكرة مما يعني نحتاج في تصميمها إلى Flipflops (حيث نتذكر أن كل قلاب هو واحدة تخزين لبت واحد).
- إذا كان لدينا n bits تخزين (n قلاب) فيمكن تمثيل 2^n حالة على الأكثر.

Next-State function:

هو تابع تركيبي combinational function حيث أنه يحدد الحالة التالية بناءً على دخل الحالة الحالية (inputs and current state).

Output function:

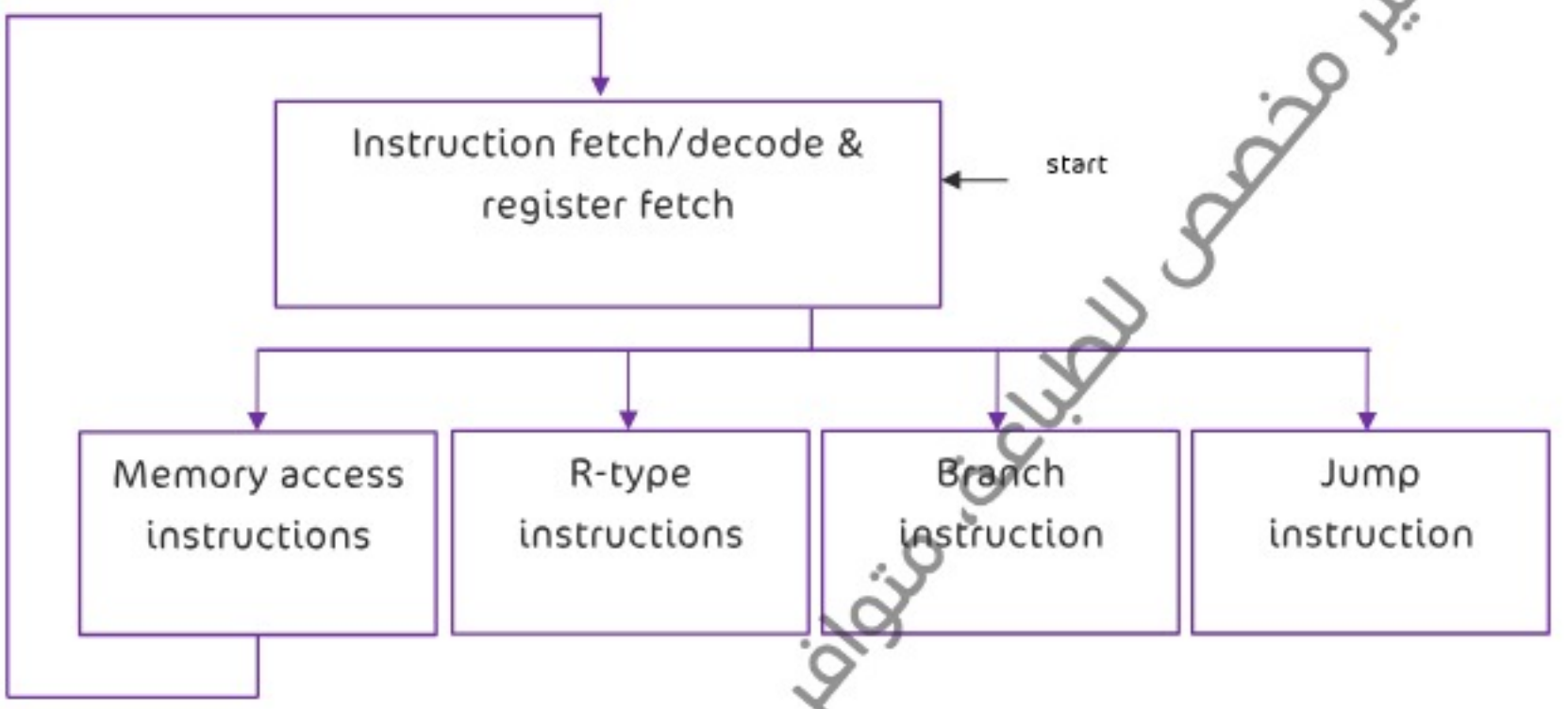
يعطي خرج حسب الحالة الحالية و الدخل.



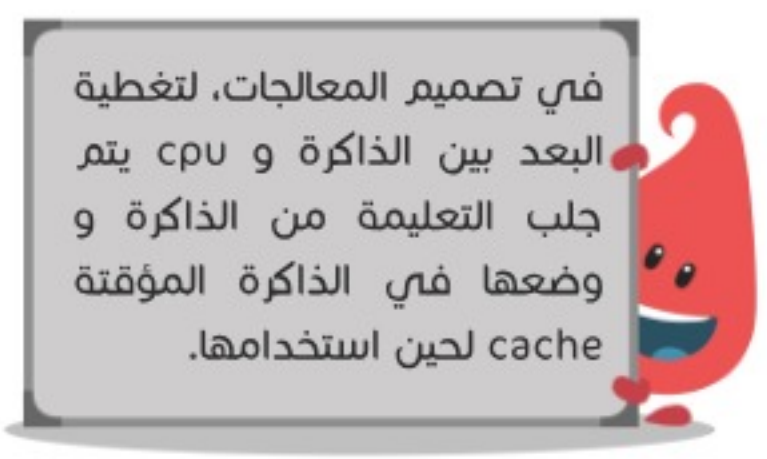
نلاحظ في تصميم FSM أعلاه:

- أن الحالة التالية تعتمد على الدخل و على الحالة الحالية، و عند نبضة الساعة التي بعدها تصبح الحالة التي كانت التالية هي الحالية و تعتمد عليها الحالة التي بعدها و هكذا...
- و إن الخرج يعتمد على الدخل و على الحالة الحالية.

High level view of FSM control for MIPS processor



- كما نعلم أن أول عملية يجب تنفيذها في الـ MIPS هي إحضار التعليمة (Fetch) حيث يجلب التعليمة من الذاكرة و وضعها في السجل الخاص بالتعليمة، التعليمة في الذاكرة تكون عبارة عن أصفار و واحدات يتم فك التشفير لها (هذه الخطوات مشتركة لكل أنواع التعليمات).
- لكن بعد فك التشفير تتم معالجة كل نوع تعليمة و إرسالها لمسارها، و هكذا يتم تنفيذ تعليمة تلو الأخرى حتى نهاية البرنامج.



Design Implementations

- عند التنفيذ باستخدام تحكم الحالة المحدودة (finite-state controller) يُحدد تابع الحالة التالية منطقياً، باستخدام ROM و PLA (Programmable Logic Arrays) و هي عبارة عن شرائح فيها العديد من البوابات المنطقية يتم وصلها لتشكيل الدارة التراكبية PLA.
- هناك طريقة بديلة لتنفيذ هذا التصميم حيث أنه يتم التحكم بتابع الحالة التالية باستخدام عدّاد (counter) مثلاً نرمز كل حالة حسب ترتيبها برقم و زيادة هذا الرقم بانتظام للانتقال إلى الحالة التي تليها.
- عندما لا يكون تتابع الحالات بشكل تسلسلي، هناك طريقة أخرى منطقية تحدد الحالة.

FSM Design steps counter based

لدينا الخطوات:

1- Encode

2- كتابة جدول الحقيقة للمنطق التركيبي.

3- استخراج العلاقات بشكلها المختصر من جدول الحقيقة

مثلاً نحتاج لترميز 10 حالات (10 تكتب بالنظام الثنائي بـ 4 بتات 1010) فنحتاج إلى 4 bits لتمثيل هذه الحالات، نكتب جدول الحقيقة للمنطق التركيبي:

(Next state table):

op code	Present state	Control signals	Next state

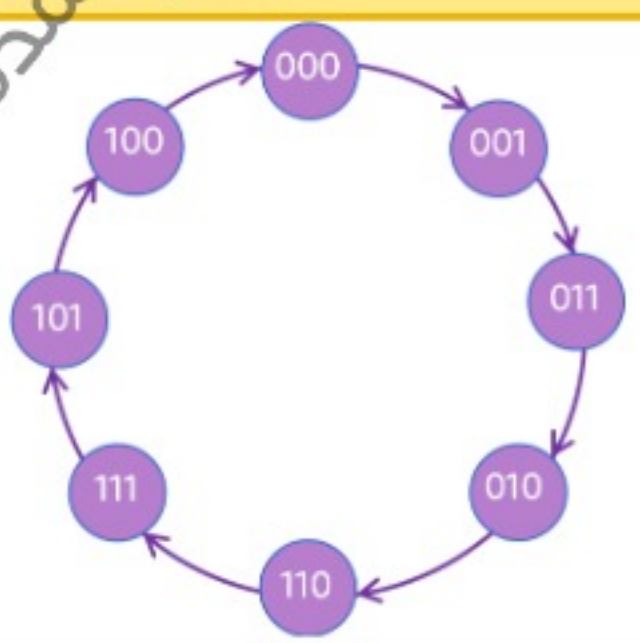
رمز التعليمة

نجمع دارة منطقية من الجدول و نستخرج العلاقة بشكلها المختصر و لأنه نحتاج لأربع بتات لتمثيل الحالة يجب أن نصل 4 قلابات بين مداخل الحالة الحالية و مخارج الحالة التالية.

↗ FSM Gray code Example:

كما نعلم تمثيل الـ Gray code يبدأ بـ 000 و ينتقل إلى الحالة التي بعدها بتغيير بت واحد فقط دون تبديل البت نفسه مرتين متتاليتين، فالحالة التالية هي 001 و التي بعدها 011 ثم 010 و هكذا...

First step: state Diagram



:State

:direction

إنه كتمثيل سلاسل كارنوف

Second step: next state table

Present state			Next state		
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0
0	0	0	0	0	1
0	0	1	0	1	1
0	1	1	0	1	0
0	1	0	1	1	0
1	1	0	1	1	1
1	1	1	1	0	1
1	0	1	1	0	0
1	0	0	0	0	0

هناك ثماني حالات أي تمثيلها بـ 3 بتات أي أننا نحتاج 3 قلابات.

Third step: Flip-Flop Transition Table

Output transition			Flip-Flops transition Inputs	
Q_N		Q_{N+1}	J	K
0	→	0	0	X
0	→	1	1	X
1	→	0	X	1
1	→	1	X	0

تذكرة بسيطة بأنواع القلابات:

- D Flip-Flops
- JK Flip-Flops
- T Flip-Flops

نستخدم الهندسة العكسية لتعبئته، أي مثلاً نقول للانتقال من الحالة 0 (الحالية) إلى الحالة التالية لتكون 1 فإذا يجب أن تكون المداخل J و K ؟ $J = 1, K = X$

Don't care

Forth step: Karnaugh tables

لنكتب جداول كارنوف لكل حالة من حالات كل قلاب، كنا قد عرفنا أننا نحتاج لثلاثة قلابات، كل قلاب له مدخلين J_i و K_i

لنبدأ بأول قلاب حيث مداخله J_1 و K_1

(1)

Present state			Next state		
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0
0	0	0	0	0	1
0	0	1	0	1	1
0	1	1	0	1	0
0	1	0	1	1	0
1	1	0	1	1	1
1	1	1	1	0	1
1	0	1	1	0	0
1	0	0	0	0	0

(2)

Q_N		Q_{N+1}	J	K
0	→	0	0	X
0	→	1	1	X
1	→	0	X	1
1	→	1	X	0

القلاب الأول:

Q_2Q_1 \ Q_0	0	1
00	1	X
01	0	X
11	1	X
10	0	X

$\overline{Q_2Q_1}$ ، Q_2Q_1

$J_0\ map$

Q_2Q_1 \ Q_0	0	1
00	X	0
01	X	1
11	X	0
10	X	1

$\overline{Q_2}Q_1$ ، $Q_2\overline{Q_1}$

$K_0\ map$

كيف يتم تعبئة الجداول؟

لدينا ثلاث قلابات (يعني ثلاث بتات) نعتبر كل Q_i حالة قلاب، نبدأ بـ Q_0 و تغيراتها (القلاب الأول).

من الجدول (1) نرى أن:

$0\ 0\ 0 = Q_2Q_1Q_0$ الحالة الحالية عند الانتقال إلى التالية تصبح $1\ 0\ 0$ نلاحظ تغير قيمة Q_0 من $0 \leftarrow 1$

نذهب إلى الجدول (2):

نرى التغير $1 \rightarrow 0$ ماذا يقابله بقيم J و K فنرى أن $J = 1\ ,K = X$ نضع هذه القيم في مقابلاتها بالجدول و هكذا لباقي القيم.

لنأخذ مثال ثاني:

الحالة $1\ 1\ 1 \leftarrow 1\ 0\ 1$ تغيرات $Q_0\ 1 \rightarrow 1$ نذهب للجدول (2) فنرى هذه الحالة يقابلها $J = X\ ,K = 0$ نضع القيم بالجداول.

و هكذا للقلابين الباقيين:

Q_2Q_1 \ Q_0	0	1
00	0	1
01	X	X
11	X	X
10	0	0

$\overline{Q_2}Q_0$

J_1

Q_2Q_1 \ Q_0	0	1
00	X	X
01	0	0
11	0	1
10	X	X

Q_2Q_0

K_1

Q_2Q_1 \ Q_0	0	1
00	0	0
01	1	0
11	X	X
10	X	X

$Q_1\overline{Q_0}$

J_2

Q_2Q_1 \ Q_0	0	1
00	X	X
01	X	X
11	0	0
10	1	0

$\overline{Q_1Q_0}$

K_2

Fifth step: Logical Expressions for flip-flops inputs.

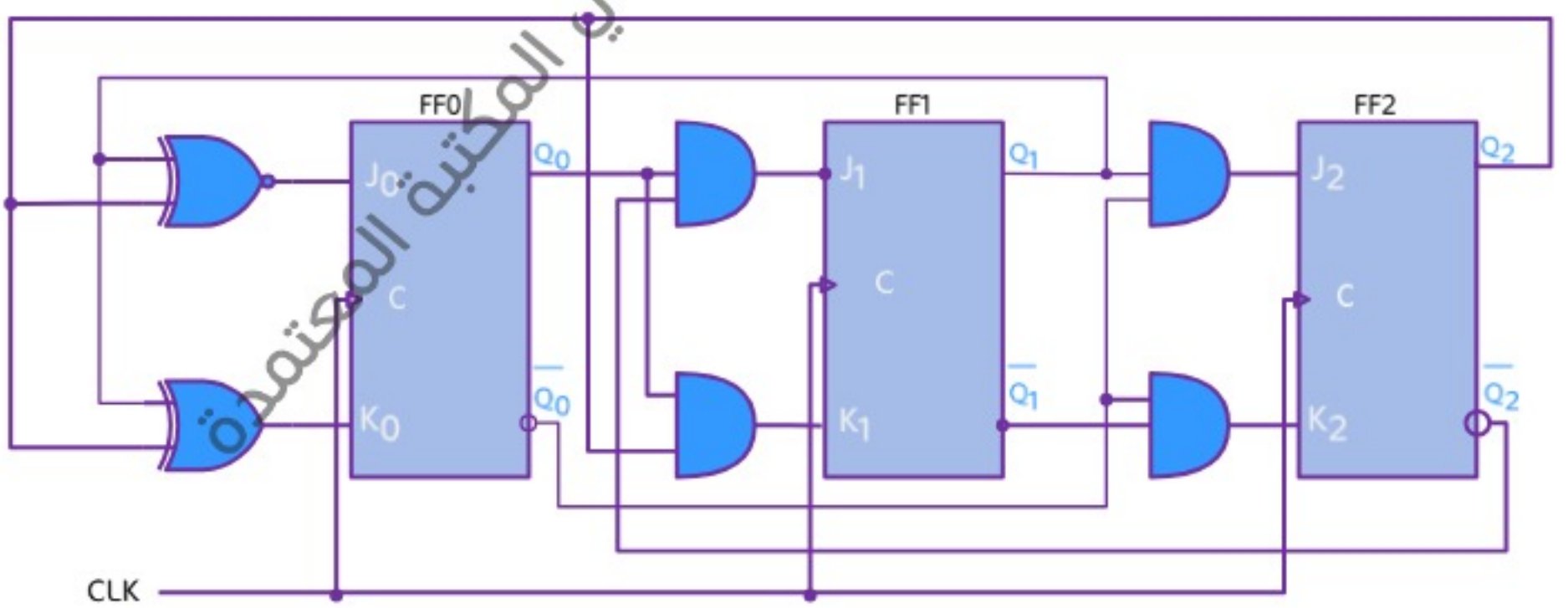
من جداول كارنوف السابقة نجد أن:

$$J_0 = Q_2Q_1 + \overline{Q_2Q_1} = \overline{Q_2 \oplus Q_1} \quad XNOR$$
$$K_0 = Q_2\overline{Q_1} + \overline{Q_2}Q_1 = Q_2 \oplus Q_1 \quad XOR$$

$$J_1 = \overline{Q_2}Q_0$$
$$K_1 = Q_2Q_0$$

$$J_2 = Q_1\overline{Q_0}$$
$$K_2 = \overline{Q_1Q_0}$$

Sixth step: counter implementation



MIPS ISA

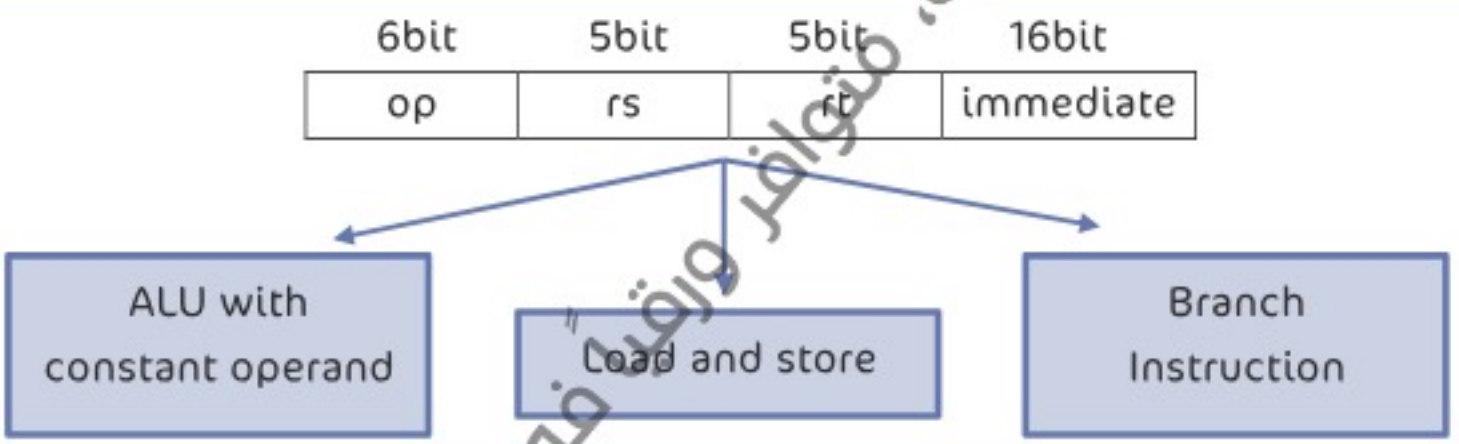
من أسهل أنواع المعالجات وأكثرهم تنظيماً هو معالج MIPS.

حجم الـ opcode ومكانه ثابت في كل أنواع التعليمات، كما نعلم يوجد ثلاثة أنواع للتعليمات وهي كالتالي:

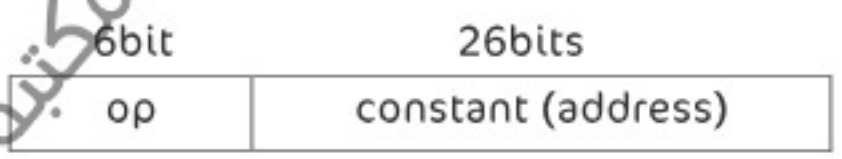
R-Type:

ALU with Register Operand

I-Type:



J-Type:



Jump Instructions

هنالك خمس أنواع عامة للتعليمات، وهي:

1- 3-operand ALU
2- ALU w/immediate
3- Loads/Stores
4- Branches
5- Jumps

تصميم وحدة التحكم الرئيسية Designing Main Control Unit

- تصميمها متعدد المستويات، حيث أنه على الرغم من بساطة معالج MIPS إلا أن تصميمها معقد، حيث أن المستوى الأساسي هو main control unit يخرج منها مستويين يعطيان خطوط تحكم، نستعمل two levels لأنه يسهل التصميم و يخفض التعقيد و يسهل الـ Debug.
- هذا التصميم لـ Single cycle.
- باستخدام التصميم متعدد المستويات:

  - سوف نمتلك وحدة تحكم لـ ALU بالإضافة إلى وحدة التحكم الرئيسية.
  - وحدة التحكم الأساسية تولد بتات ترميز الـ ALU التي يتم استخدامها كدخل لوحدة التحكم للـ ALU التي تولد الإشارات الفعلية التي تتحكم بوحدة الحساب والمنطق (هذه التقنية الشائعة للتنفيذ).

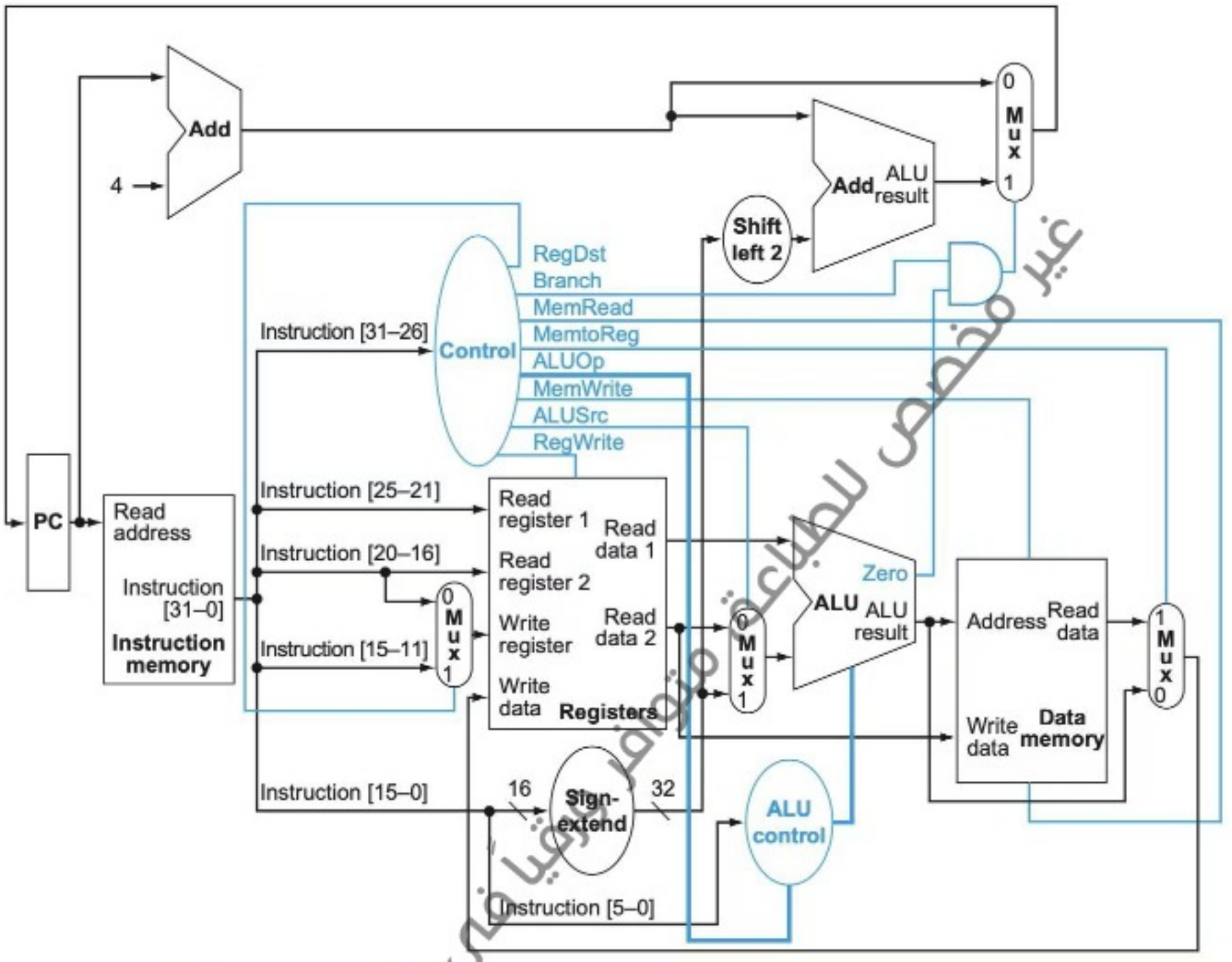
- باستخدام تقنية تعدد مستويات التحكم يمكننا تقليل حجم وحدة التحكم الاساسية.

لنتذكر معنى:

- **Single cycle**: أي أن كل تعليمة تأخذ نبضة ساعة واحدة.
- **Multi cycle**: أي أن كل تعليمة تستغرق أكثر من نبضة ساعة.
- و لكن Multi cycle أفضل من single cycle و لكن أصعب بالتحكم لأنه يختلف عدد نبضات الساعة من تعليمة لأخرى.

العالم بانتظار أفكارك..

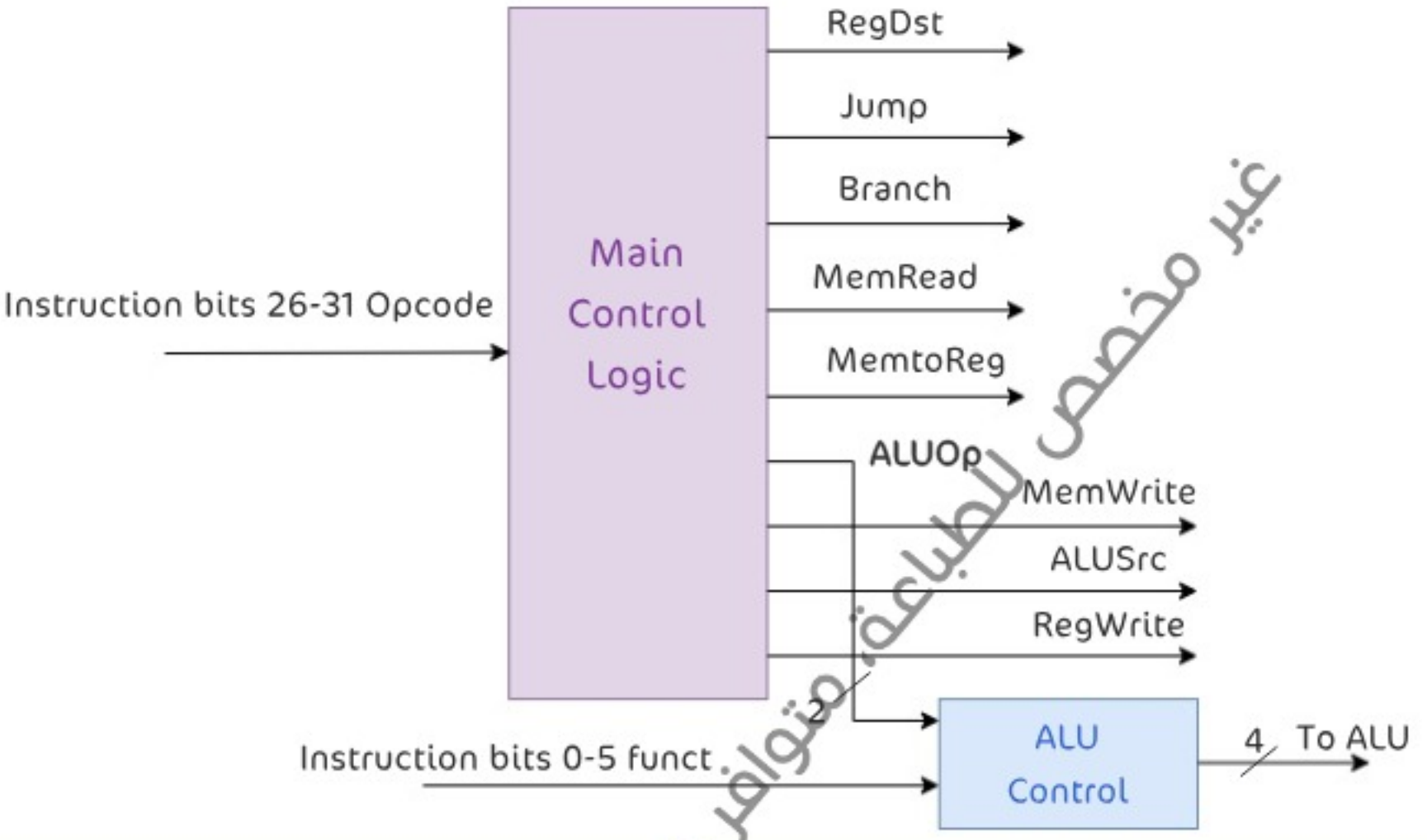
Combined Data paths and control lines (single cycle)



- الـ Mux (multiplexer) في المخطط أعلاه يتحكم بالتعليمة مثلاً إذا كانت add أو addi.
- و كما نلاحظ أعلاه أن المتحكم أعطى ALUOp و أخذها إلى ALU control لعمل ضرب أو جمع أو قسمة...

و المخطط التالي يوضح هذه العملية:

Designing the main control unit (Single Cycle):



Opcode و funct يساعدون في معرفة نوع التعليمة.

↙ Example: Designing ALU control unit

ليكن لدينا الستة توابع التالية:

جميع التعليمات السابقة من نمط R-Type

و تعليمة add تستخدم أيضاً لعمليات الذاكرة مثل: load/store

و تعليمة subtract تستخدم في القفز المشروط (Branch)

و مما سبق نستنتج أن عدد حالات الـ FSM هو 3:

1. R-Type (with funct field)
2. Load/Store
3. Branch (beq)

ALU control lines	Function
0000	AND
0001	OR
0010	Add
0110	Subtract
0111	Set on less than (slt)
1100	NOR

نحتاج لترميز هذه الحالات الثلاثة لبتين (2 bits) حيث $2^2 = 4$ والحالة الرابعة لا نستخدمها.

- نستطيع أن نولّد مدخلات التحكم ALU ذات الـ 4bits باستخدام وحدة تحكم صغيرة و هي: ALU control unit.
- تحتوي كمدخلات على حقل function التعليمة و حقل تحكم من 2bit و الذي نسميه ALUOp.
- ALUOp يشير فيما إذا كانت العملية التي ستنفذ:

- add (00) لعمليتي load و store.
- subtract (01) لعملية beq.
- R-Type (10) تعرف بالعملية المرمزة في حقل function.
- الحالة (11) غير مستخدمة (هي الحالة الرابعة و لا يوجد لدينا إلا ثلاث حالات).

- خرج وحدة تحكم الـ ALU هي أربع بتات تتحكم مباشرة بـ ALU بتوليد إحدى المجموعات المؤلفة من البتات الأربعة.

Instruction Opcode	ALUOp	Instruction Operation	Funct field	Desired ALU action	ALU control input
LW	00	Load word	XXXXXX	Add	0010
SW	00	Store word	XXXXXX	Add	0010
Branch equal	01	Branch equal	XXXXXX	Subtract	0110
R-Type	10	Add	100000	Add	0010
R-Type	10	Subtract	100010	Subtract	0110
R-Type	10	AND	100100	AND	0000
R-Type	10	OR	100101	OR	0001
R-Type	10	Set on less than	101010	Set on less than	0111

نعلم أنه لا يوجد حقل funct إلا في R-Type البتين الأعلى من حقل funct هما الـ ALUOp، و يتم التغير في باقي البتات حسب نوع التعليمة.

ALUOp		Funct field						Operation
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	
0	0	X	X	X	X	X	X	0010
X	1	X	X	X	X	X	X	0110
1	X	X	X	0	0	0	0	0010
1	X	X	X	0	0	1	0	0110
1	X	X	X	0	1	0	0	0000
1	X	X	X	0	1	0	1	0001
1	X	X	X	1	0	1	0	0111

لم يتم استخدام الحالة 11 لذلك وضعنا X1 عوضاً عن 01 و وضعنا 1X عوضاً عن 10.

عندما يكون الـ funct field مستخدم يكون البتين F4 و F5 10 دوماً لأنه يوجد فقط في R-Type، و لذلك وضعنا مكان هذين البتين X X (don't care) في جدول الحقيقة.

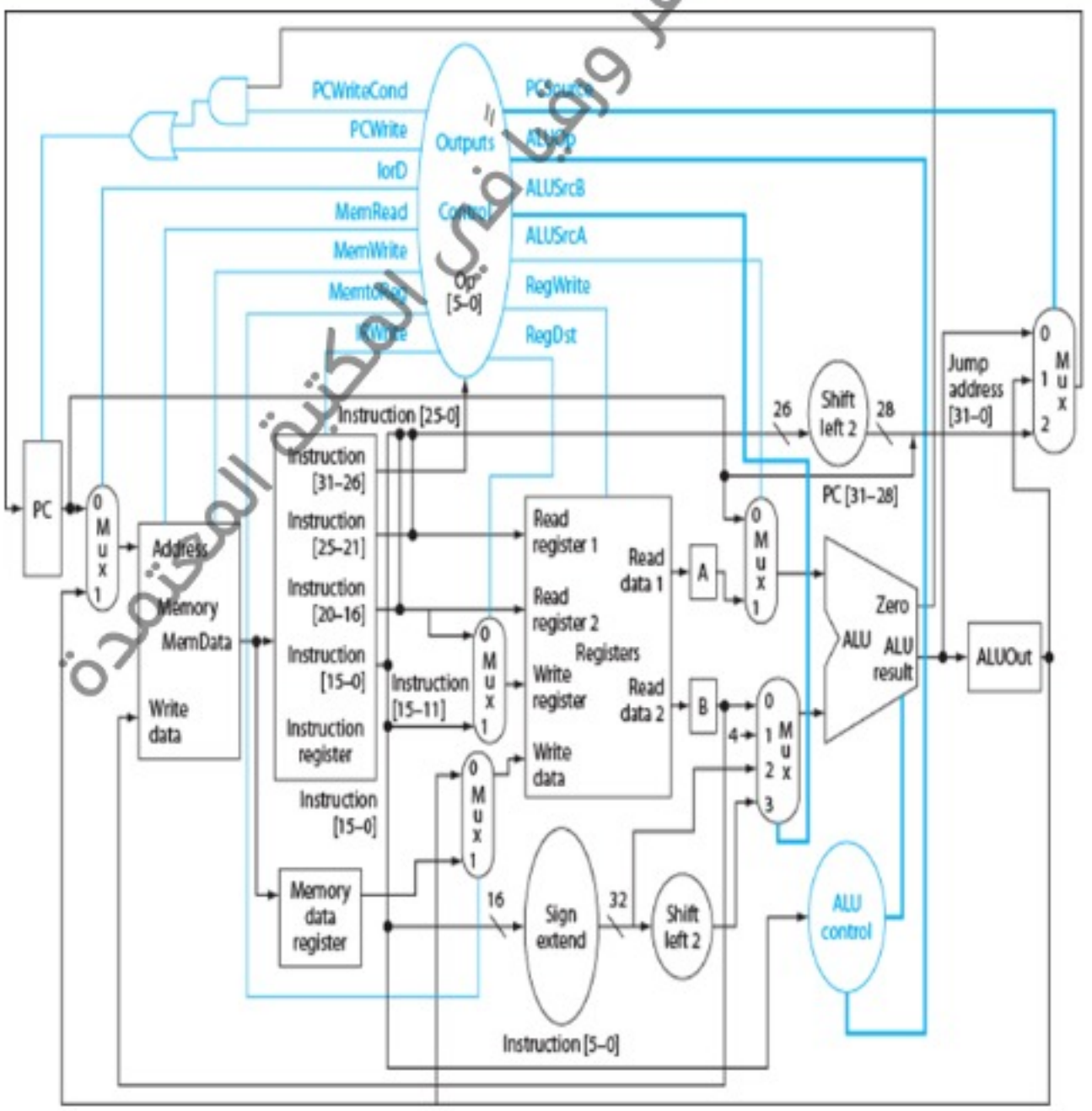
Designing the main control unit

يوجد العديد من الملاحظات المختلفة عن بنية تعليمات MIPS:

- حقل الـ Opcode دائماً ثابت الموقع والحجم حيث موقعه البتات $26 \leftarrow 31$ و هي البتات الستة الأخيرة اليسارية.
- السجلان اللذان تتم قراءتهما مخصصان و هما السجلان $rs : 21 \leftarrow 25$ البت و السجل $rt : 16 \leftarrow 20$، و هذا ينطبق على التعليمات من النوع R-Type, branch equal, store Instructions.
- السجل الأساسي للتخزين والتحميل هو rs.
- الـ 16 بت المخصصين لتعليمات store-load-branch (غالباً يدلون على العنوان) دائماً يكون موقعهم في البتات $0 \leftarrow 15$ (أقصى اليمين).
- سجل الوجهة يكون إحدى المكانين:

للتحميل (تعليمة load) تكون في السجل rt الذي موقعه البتات $16 \leftarrow 20$.

التعليمات من نوع R-Type في السجل rd الذي موقعه البتات $11 \leftarrow 15$ نحتاج ناخب (Mux) لاختيار حقل التعليمة المستخدم.



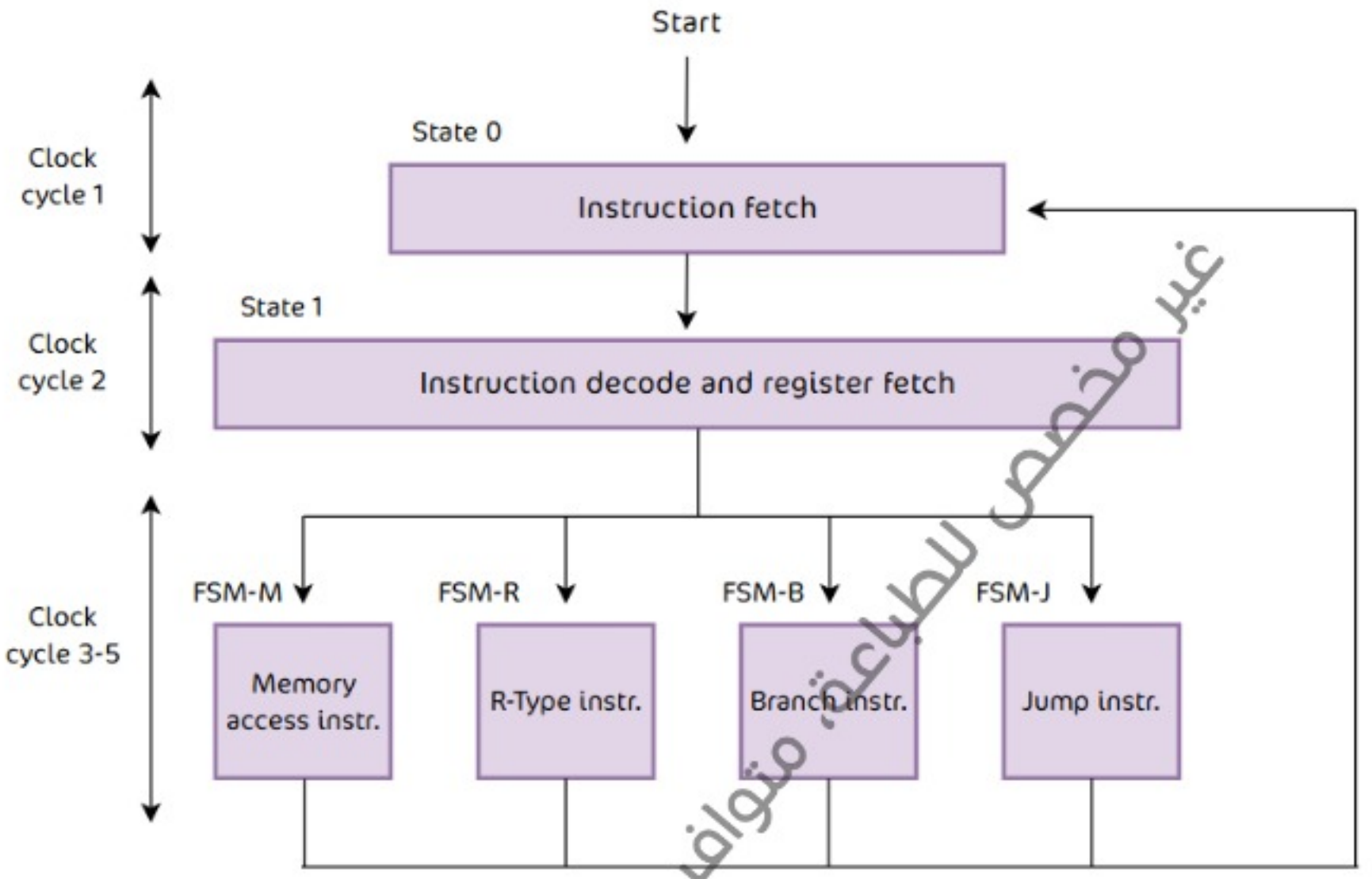
Multi-Cycle

- نلاحظ من المخطط أعلاه وجود ذاكرة واحدة فقط.
- و لم نحتاج إلى ALU ثانية لزيادة عداد الـ pc مثل تصميم الـ single-cycle حيث أنها مثلاً تجمع عدد (تنفذ تعليمة) نبضة و ويعود التنفيذ إليها بنبضة ثانية لزيادة الـ pc.

3 to 5 Cycles for an Instruction

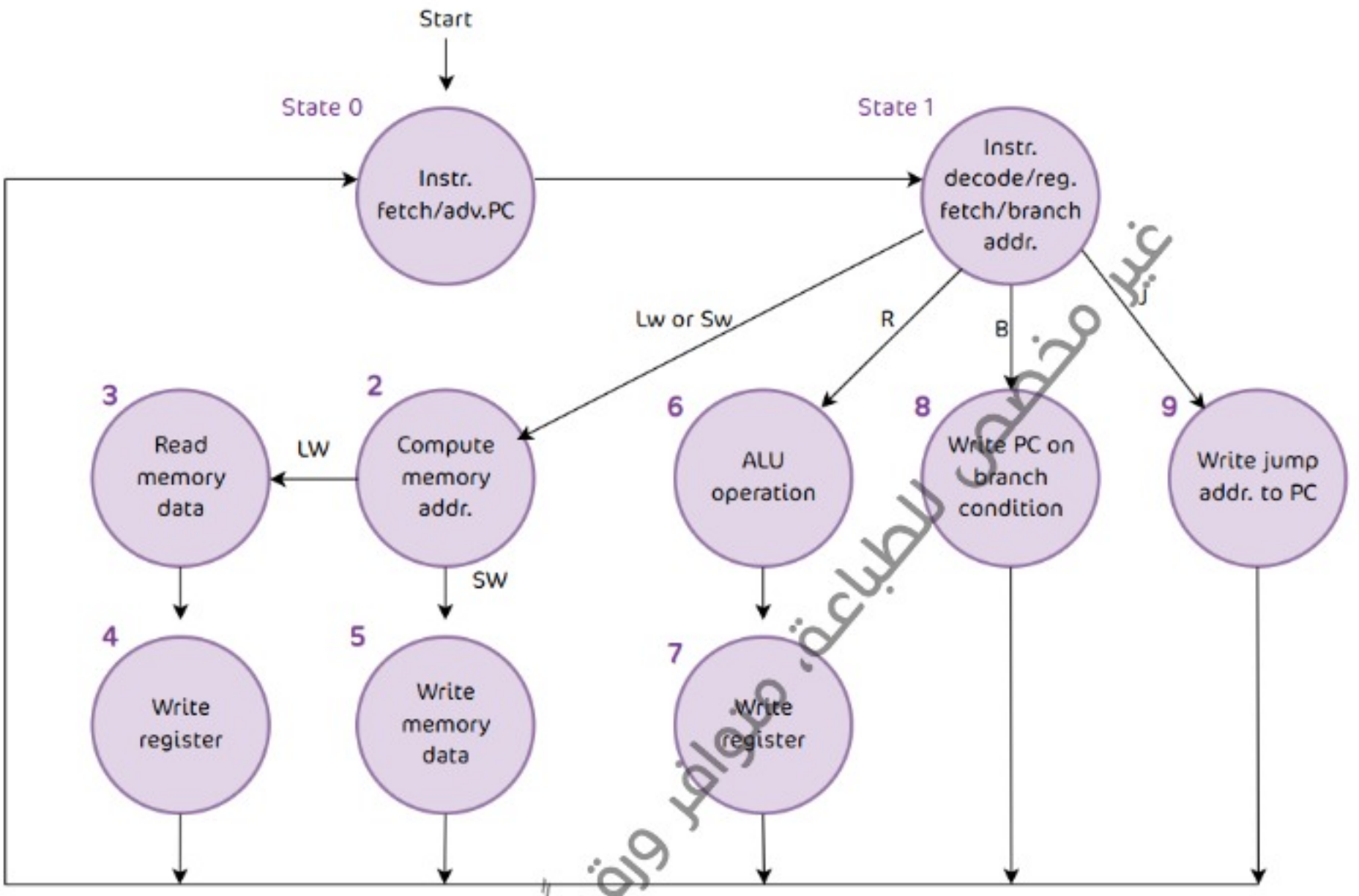
Step	R-Type (4 cycle)	Mem.Ref. (4 or 5 cycle)	Branch type (3 cycles)	J-Type (3 cycles)
Instruction fetch	IR←Memory [PC]; PC←PC+4			
Instr.decode/ Reg.fetch	A←Reg(IR[21-25]); B←Reg(IR[16-20]) ALUOut←PC+(sign extend IR [0-15]) <<2			
Execution, addr. Comp., branch & jump completion	ALUOut←A op B	ALUOut←A+sign extend (IR[0-15])	If(A==B) then PC←ALUOut	PC←PC[28-31] ‖ (IR[0-25]<<2)
Mem.Access or R-type completion	Reg(IR[11-15]) ←ALUOut	MDR←M[ALUOut] or M[ALUOut] ←B		
Memory Read completion		Reg(IR[16-20]) ←MDR		

Control: Finite State Machine



- أقصر تعليمة هي تعليمة jump حيث CPI=3
- أطول تعليمة هي تعليمة load حيث CPI=5، لأنه يتطلب الذهاب للذاكرة و جلب البيانات و إعادة تحميلها في السجلات.

Multi-Cycle FSM Control



- المداخل: ستة بتات opcode
- المخارج: 16 إشارة تحكم
- نريد معرفة عدد دورات تعليمة ما من هذا المخطط

مثلاً: تعليمة load تمر بـ state 0 و state 1 و state 2 و state 3 و state 4 ← 5 نبضات ساعة.

The End.