دانشگاه کردستان
University of Kurdistan
زانکۆی کوردستان

Department of Computer Engineering
University of Kurdistan

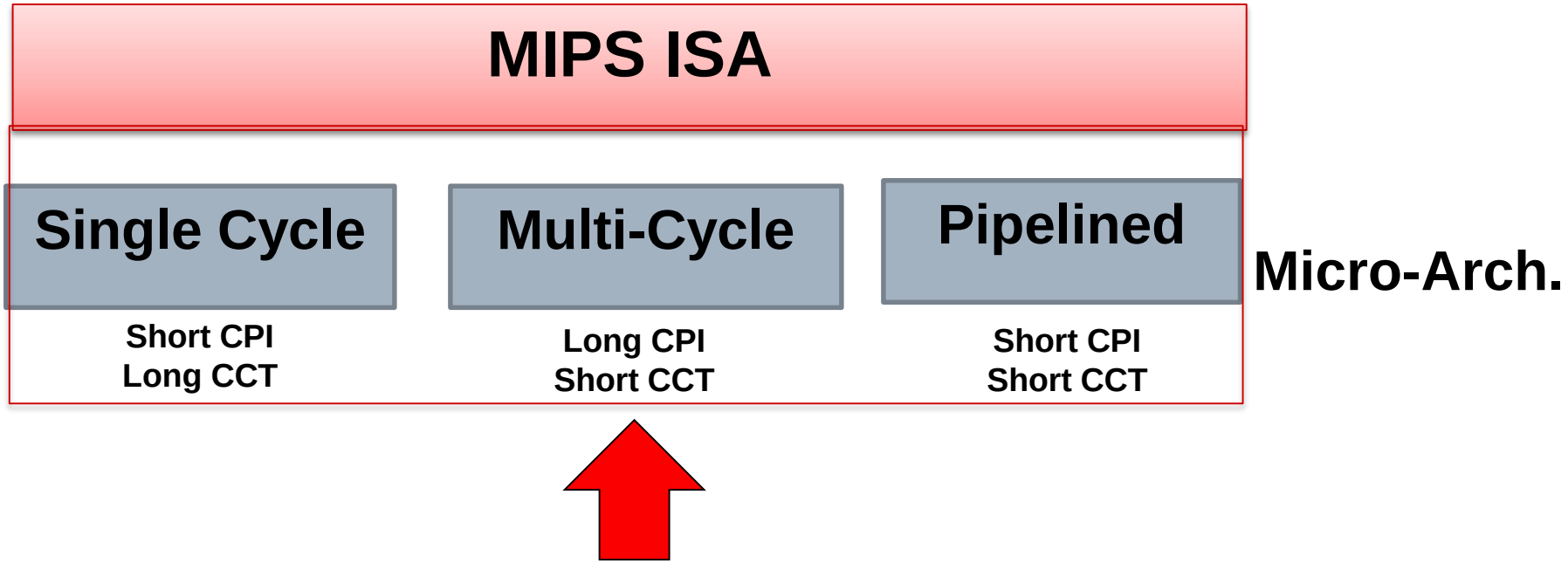
Computer Architecture

MIPS Multi-cycle Implementation

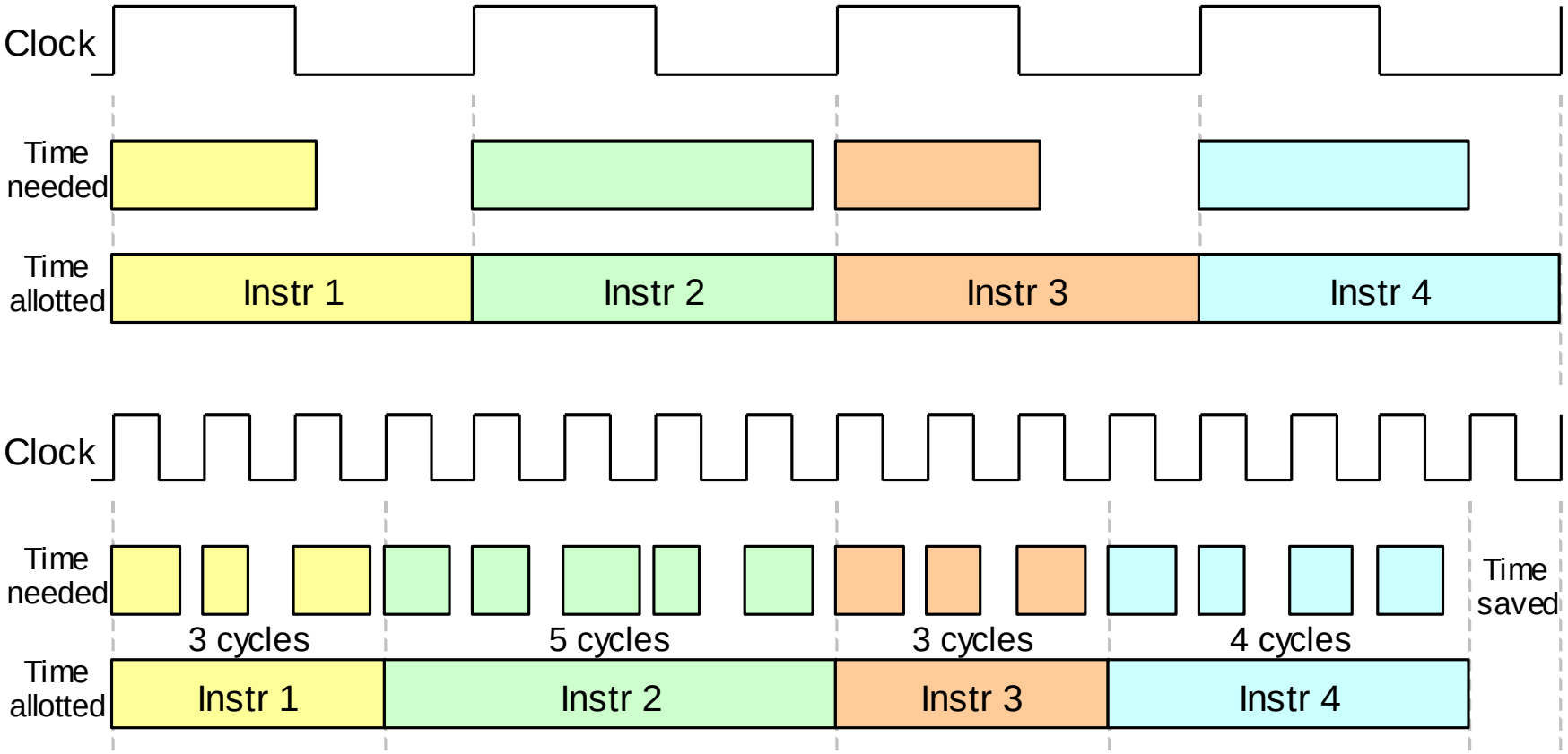
By: Dr. Alireza Abdollahpouri

A Multi-cycle MIPS processor

Any instruction set can be implemented in many different ways



A Multicycle Implementation

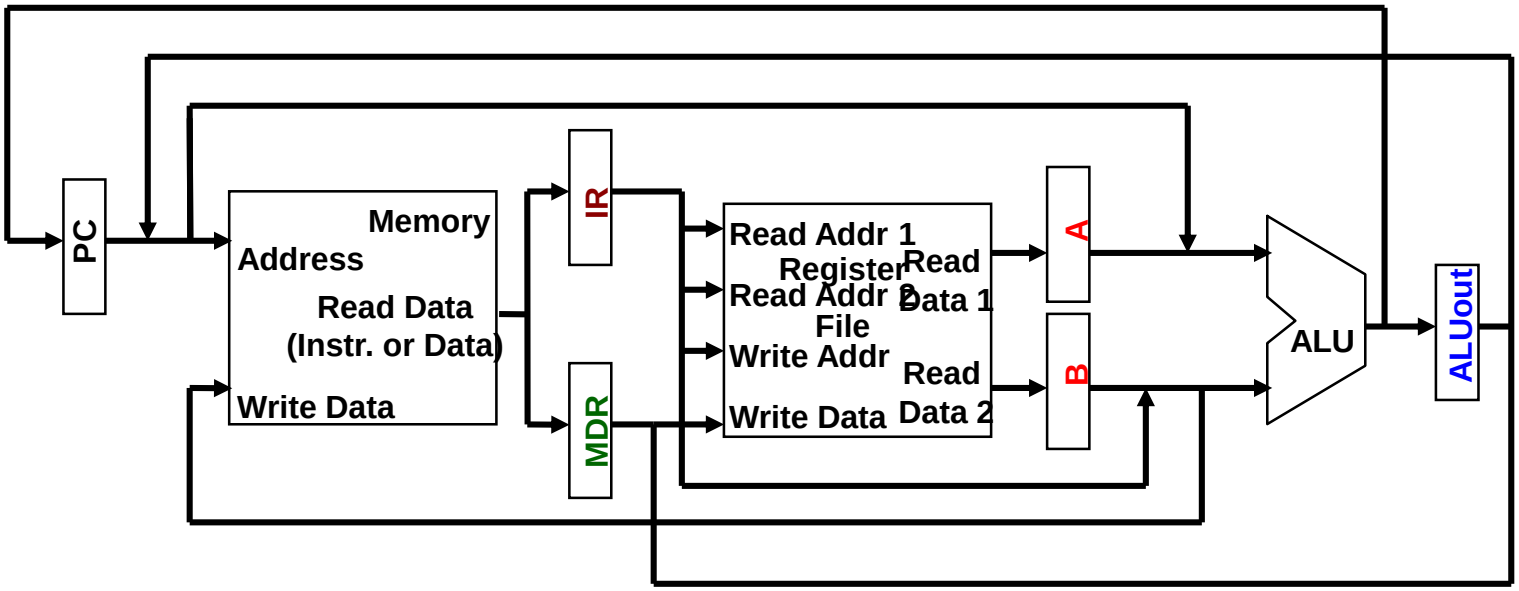


Single-cycle versus multicycle instruction execution.

نگرش Multicycle Datapath

- هر دستور به تعدادی مرحله کوچکتر تقسیم شده و هر یک از این مراحل در یک کلاک اجرا میشوند. بدین ترتیب برای اجرای هر دستور به تعدادی کلاک کوچک تر نیاز خواهیم داشت.
- مراحل طوری انتخاب میشوند که کار انجام گرفته در آنها متعادل باشد.
- در هر مرحله فقط از یکی از بلوکهای سخت افزاری اصلی استفاده میشود.
- هر دستور تعداد متفاوتی کلاک لازم دارد.
- فقط به یک حافظه نیاز دارد. البته در هر سیکل فقط میتوان یکبار به حافظه دسترسی داشت.
- فقط به یک ALU/adder نیاز دارد. البته در هر سیکل بیش از یکبار از ALU نمیتوان استفاده نمود.

Multicycle Datapath نگرش



در این معماری مقادیری که در سیکلهای بعدی دستور مورد نیاز هستند در رجیسترهائی ذخیره میشوند. در نتیجه باید اجزای زیر به معماری افزوده شوند:

MDR – Memory Data Register

IR – Instruction Register

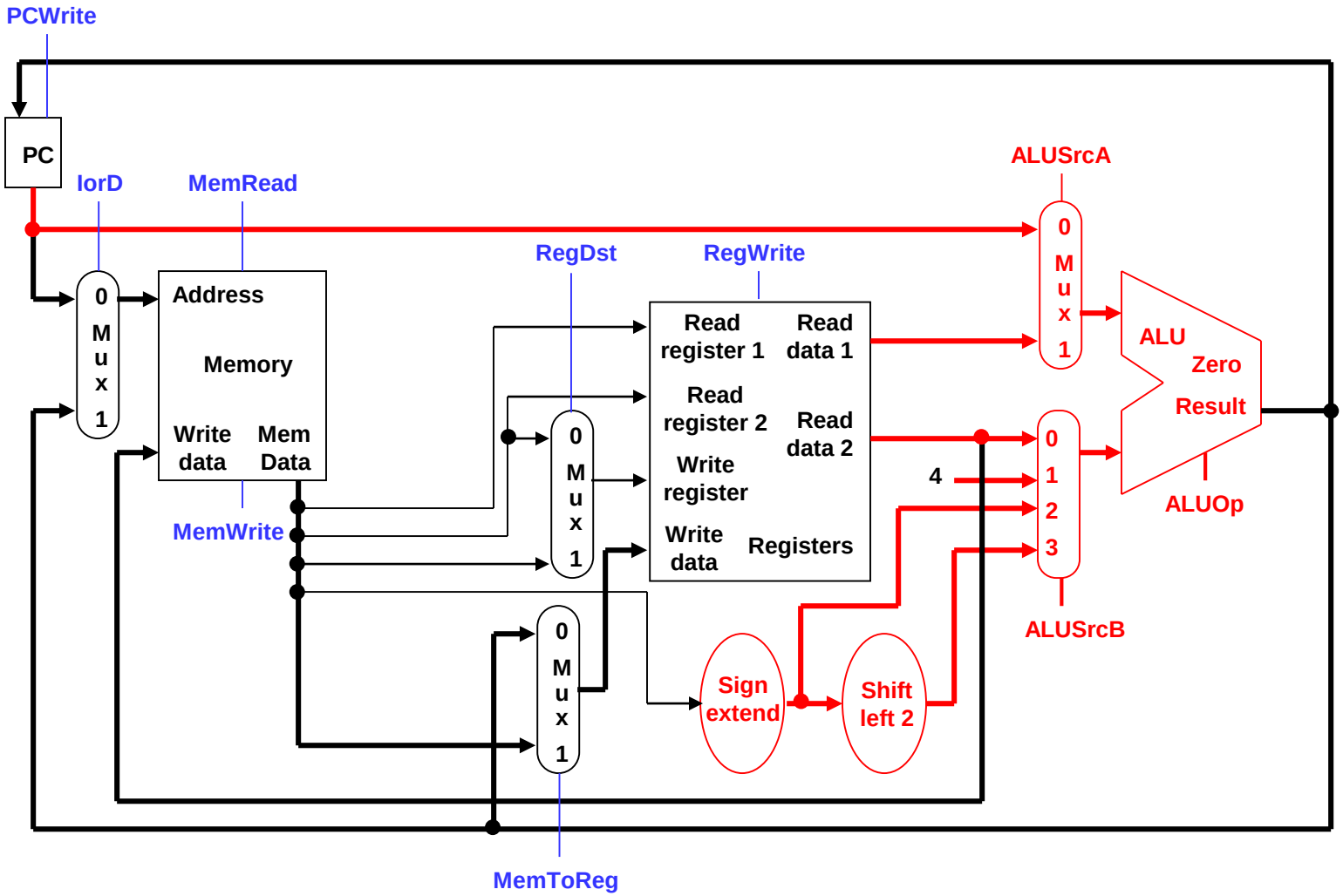
ALUout – ALU output register

A, B – regfile read data registers

Our new adder setup

- We can eliminate *both* extra adders in a multicycle datapath, and instead use just one ALU, with multiplexers to select the proper inputs.
- A 2-to-1 mux ALUSrcA sets the first ALU input to be the PC or a register.
- A 4-to-1 mux ALUSrcB selects the second ALU input from among:
  - the register file (for arithmetic operations),
  - a constant 4 (to increment the PC),
  - a sign-extended constant (for effective addresses), and
  - a sign-extended and shifted constant (for branch targets).
- This permits a single ALU to perform all of the necessary functions.
  - Arithmetic operations on two register operands.
  - Incrementing the PC.
  - Computing effective addresses for lw and sw.
  - Adding a sign-extended, shifted offset to (PC + 4) for branches.

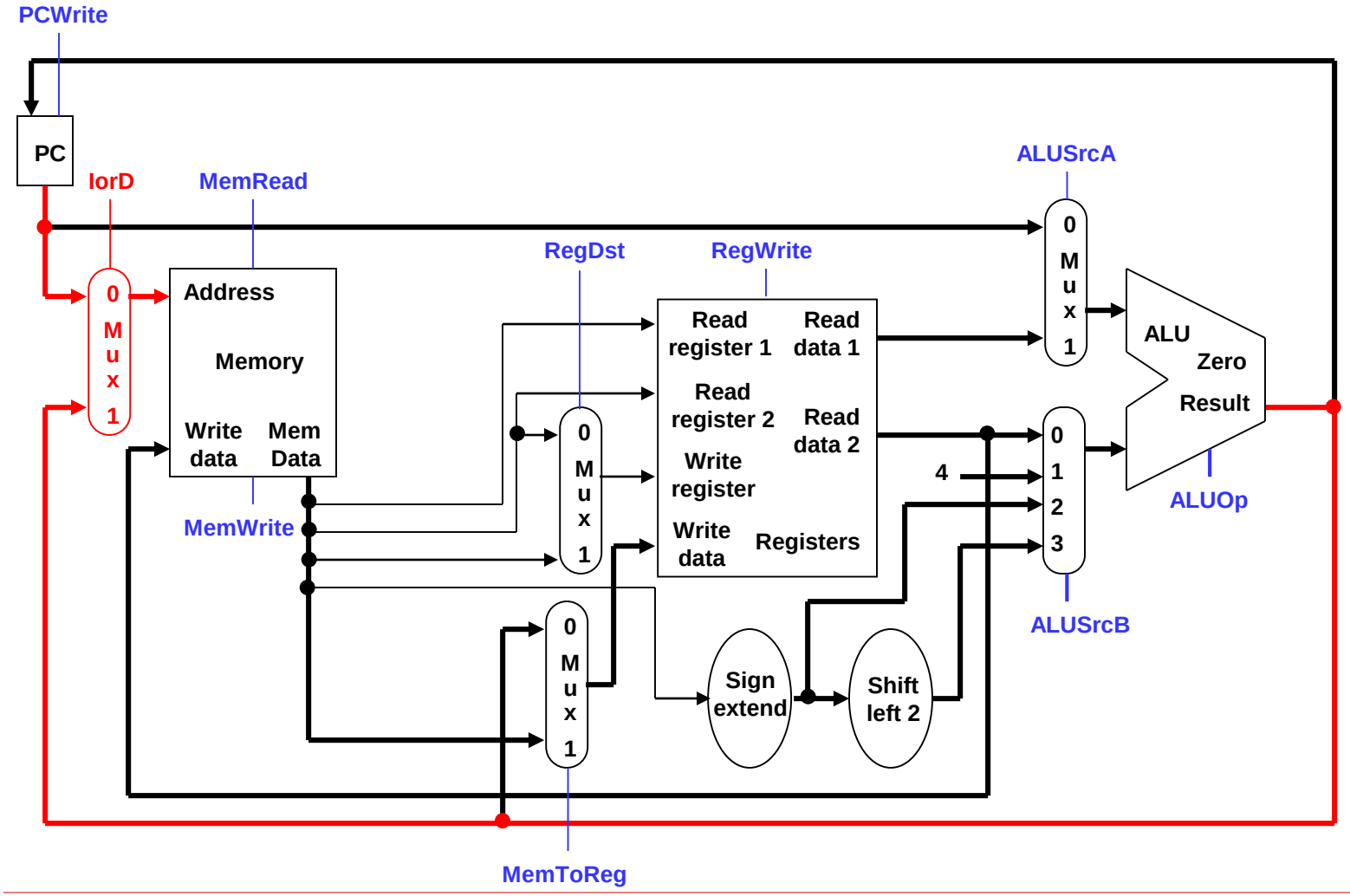
The multicycle adder setup highlighted



Eliminating a memory

- Similarly, we can get by with one unified memory, which will store *both* program instructions *and* data. (a Princeton architecture)
- This memory is used in both the instruction fetch and data access stages, and the address could come from either:
  - the PC register (when we're fetching an instruction), or
  - the ALU output (for the effective address of a lw or sw).
- We add another 2-to-1 mux, IorD, to decide whether the memory is being accessed for instructions or for data.

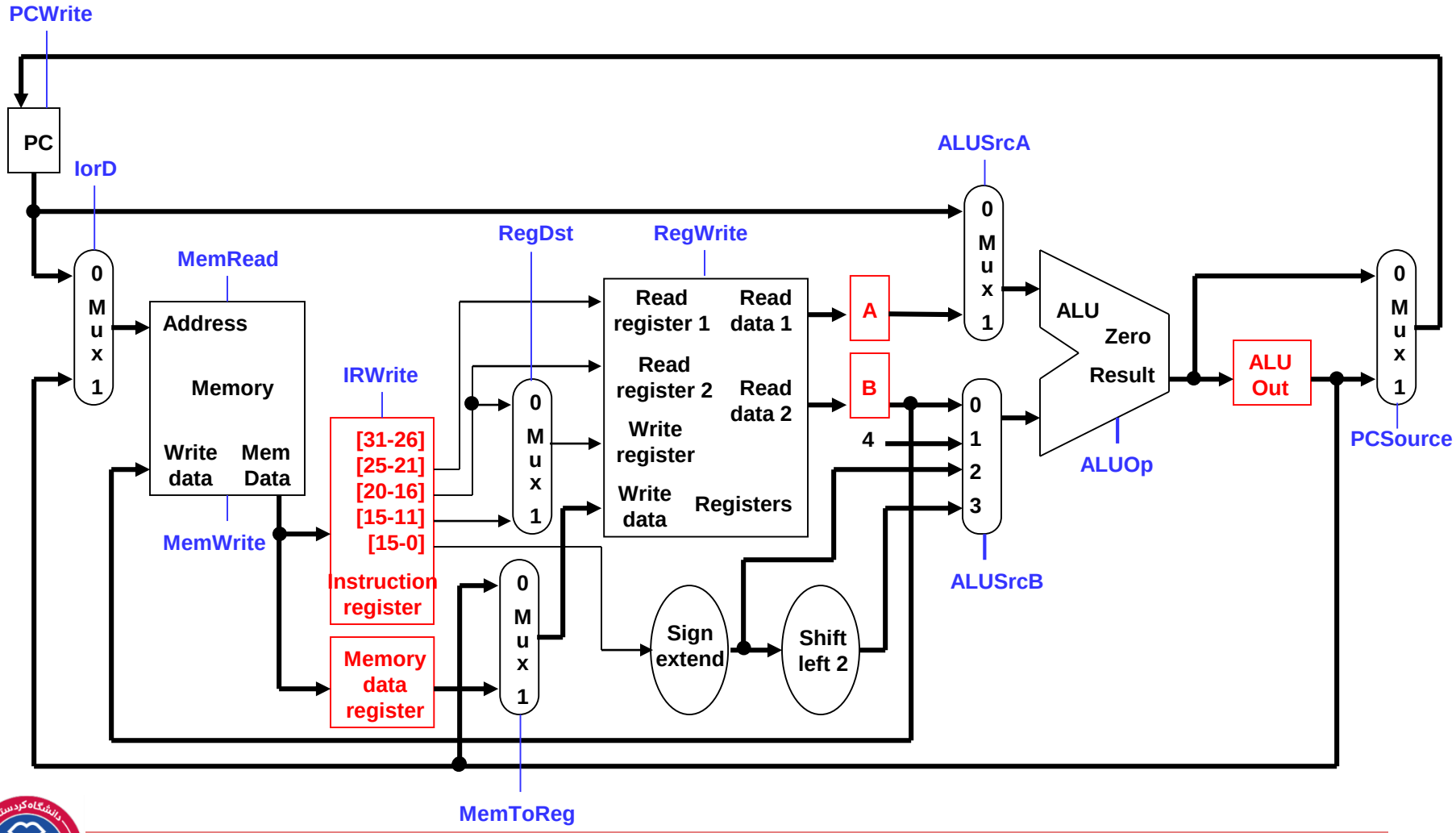
The new memory setup highlighted



Intermediate registers

- Sometimes we need the output of a functional unit in a later clock cycle during the execution of one instruction.
  - The instruction word fetched in stage 1 determines the destination of the register write in stage 5.
  - The ALU result for an address computation in stage 3 is needed as the memory address for lw or sw in stage 4.
- These outputs will have to be stored in intermediate registers for future use. Otherwise they would probably be lost by the next clock cycle.
  - The instruction read in stage 1 is saved in Instruction register.
  - Register file outputs from stage 2 are saved in registers A and B.
  - The ALU output will be stored in a register ALUOut.
  - Any data fetched from memory in stage 4 is kept in the Memory data register, also called MDR.

The final multicycle datapath

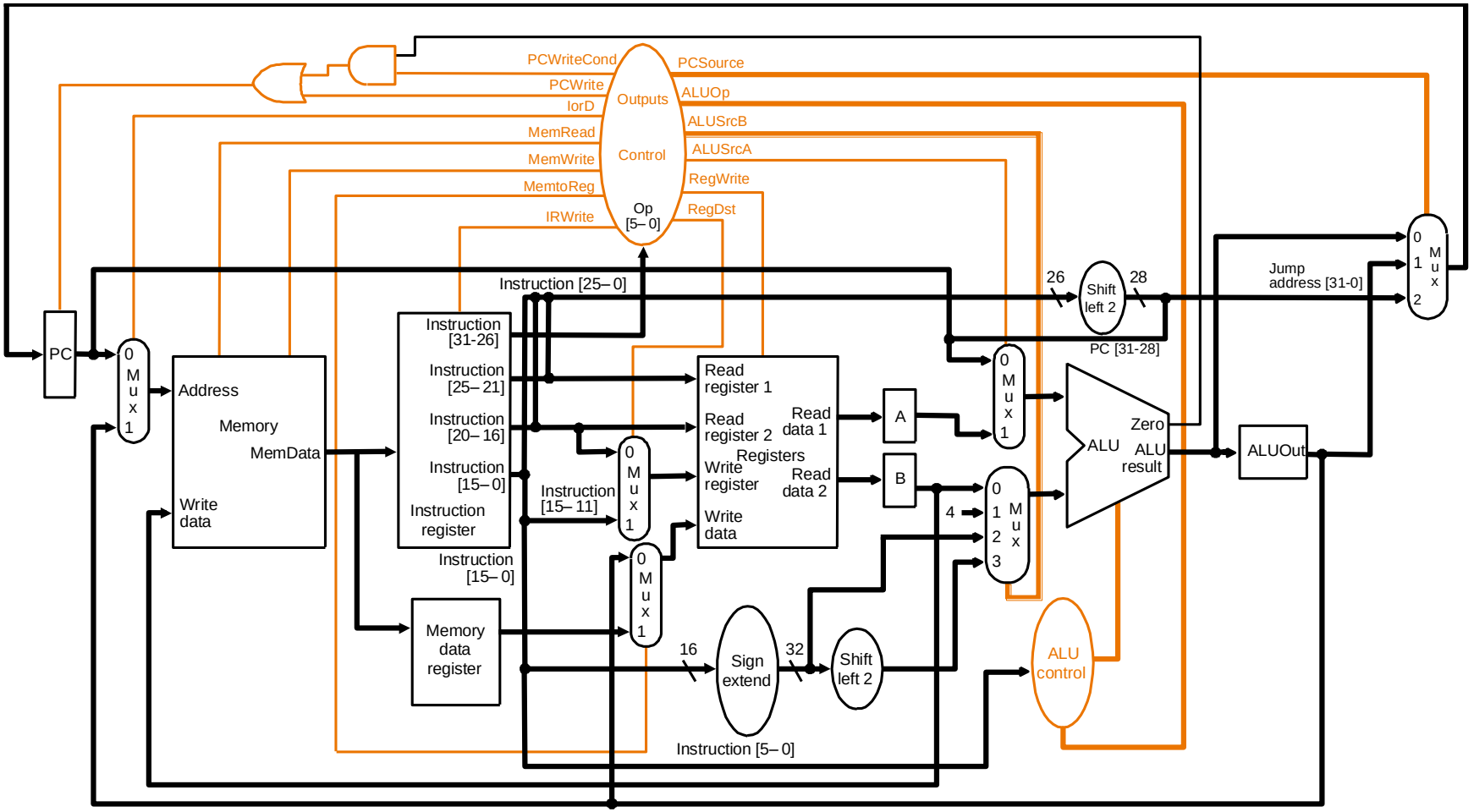


مسیر داده چند سیکله

مسیر داده چند سیکله سه تفاوت عمده با مسیرداده تک سیکله دارد:

۱- حافظه برنامه و داده ادغام شده است.
۲- تعدادی رجیستر برای ذخیره داده های میانی اضافه شده است.
۳- واحد ALU وظیفه مدارات جمع کننده (برای مقصد پرش و محاسبه آدرس بعدی) را نیز انجام میدهد.

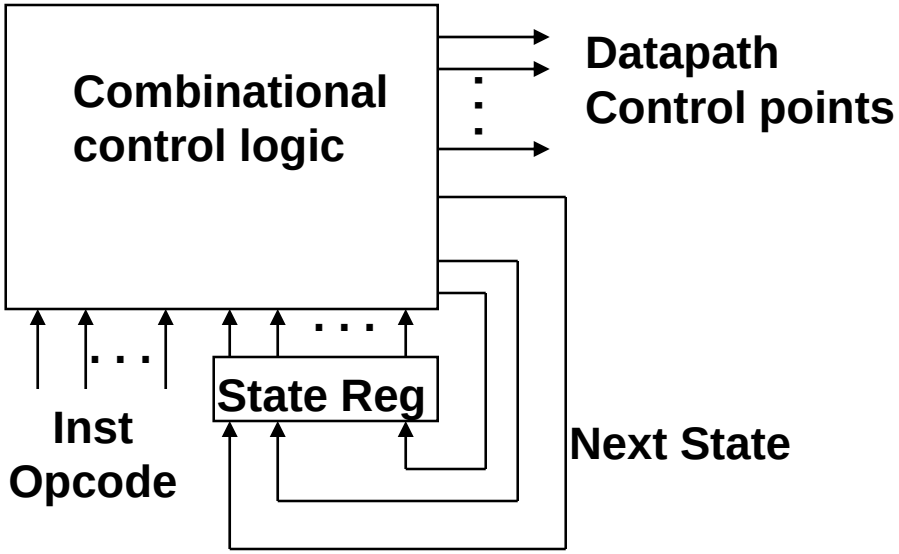
Multicycle Datapath with Control



Multicycle control unit

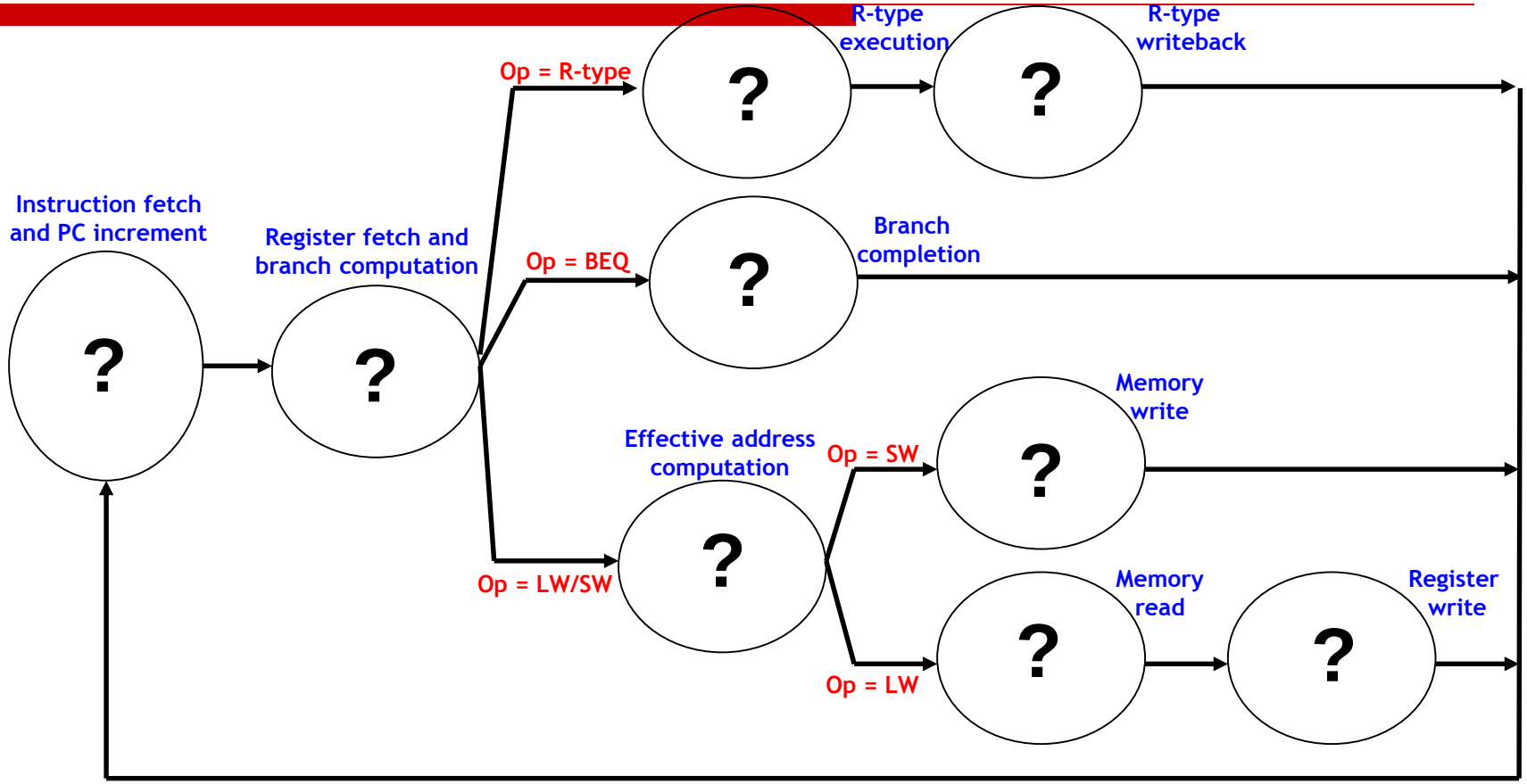
- The control unit is responsible for producing all of the control signals.
- Each instruction requires a *sequence* of control signals, generated over multiple clock cycles.
  - This implies that we need a state machine.
  - The datapath control signals will be outputs of the state machine.
- Different instructions require different sequences of steps.
  - This implies the instruction word is an input to the state machine.
  - The next state depends upon the exact instruction being executed.
- After we finish executing one instruction, we'll have to repeat the entire process again to execute the next instruction.

واحد کنترل Multicycle



- در معماری Multicycle سیگنالهای کنترل را نمیتوان فقط از روی بیت های دستورالعمل بدست آورد.
- از اینرو از یک ماشین FSM برای طراحی واحد کنترل استفاده میشود.
  - تعدادی state محدود برای پردازنده فرض میشود که در state reg ذخیره میشوند.
  - state بعدی از روی state فعلی ومقادیر ورودی تعیین میشوند.

Finite-state machine for the control unit



- **Each bubble is a state**
  - **Holds the control signals for a single cycle**
  - **Note: All instructions do the same things during the first two cycles**

Stage 1: Instruction Fetch

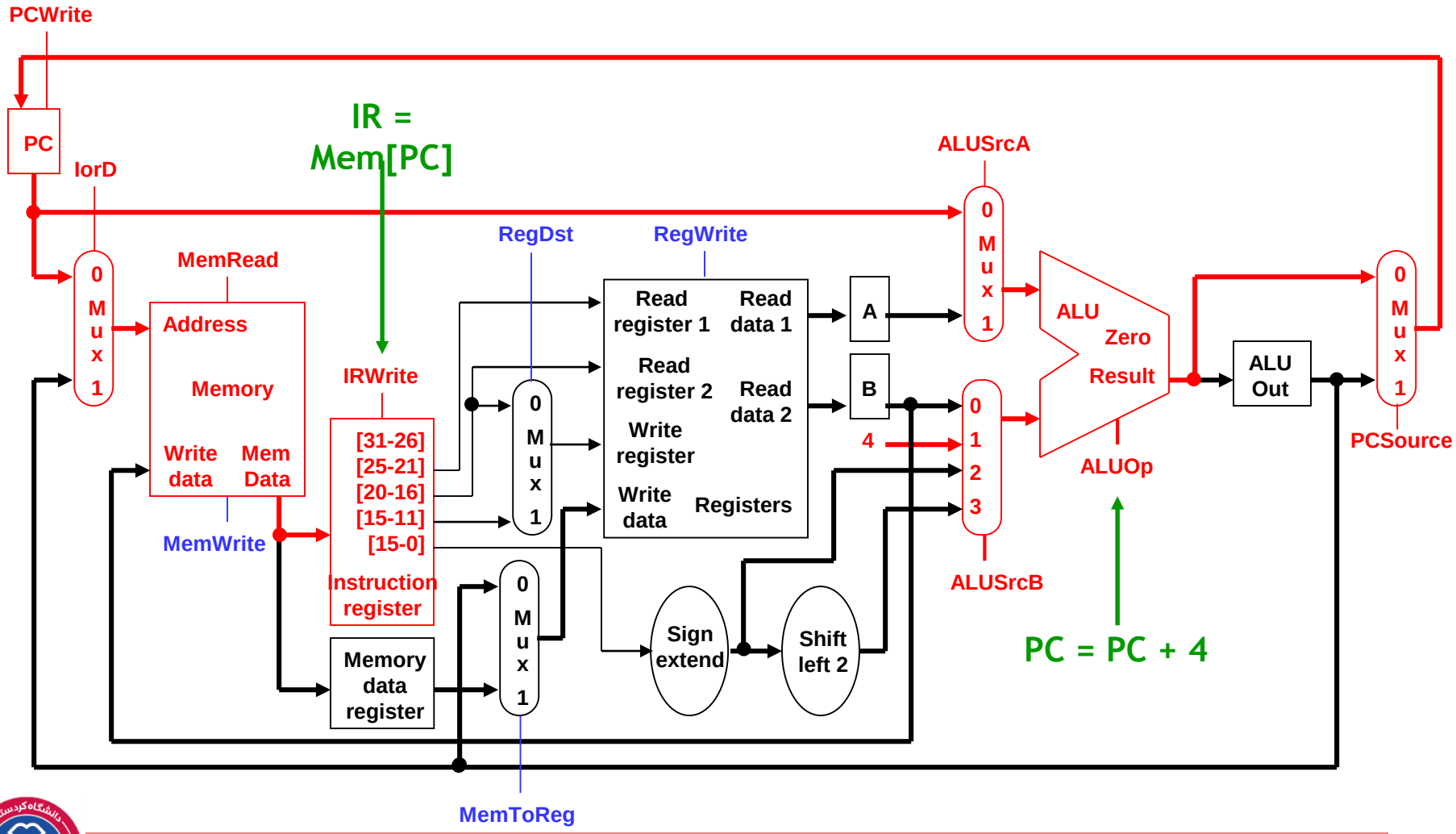
- Stage 1 includes two actions which use two separate functional units: the memory and the ALU.
  - Fetch the instruction from memory and store it in IR.

$$IR = Mem[PC]$$

  - Use the ALU to increment the PC by 4.

$$PC = PC + 4$$

Stage 1: Instruction fetch and PC increment



Stage 1 control signals

- Instruction fetch: IR = Mem[PC]

Signal	Value	Description
MemRead	1	Read from memory
IorD	0	Use PC as the memory read address
IRWrite	1	Save memory contents to instruction register

- Increment the PC: PC = PC + 4

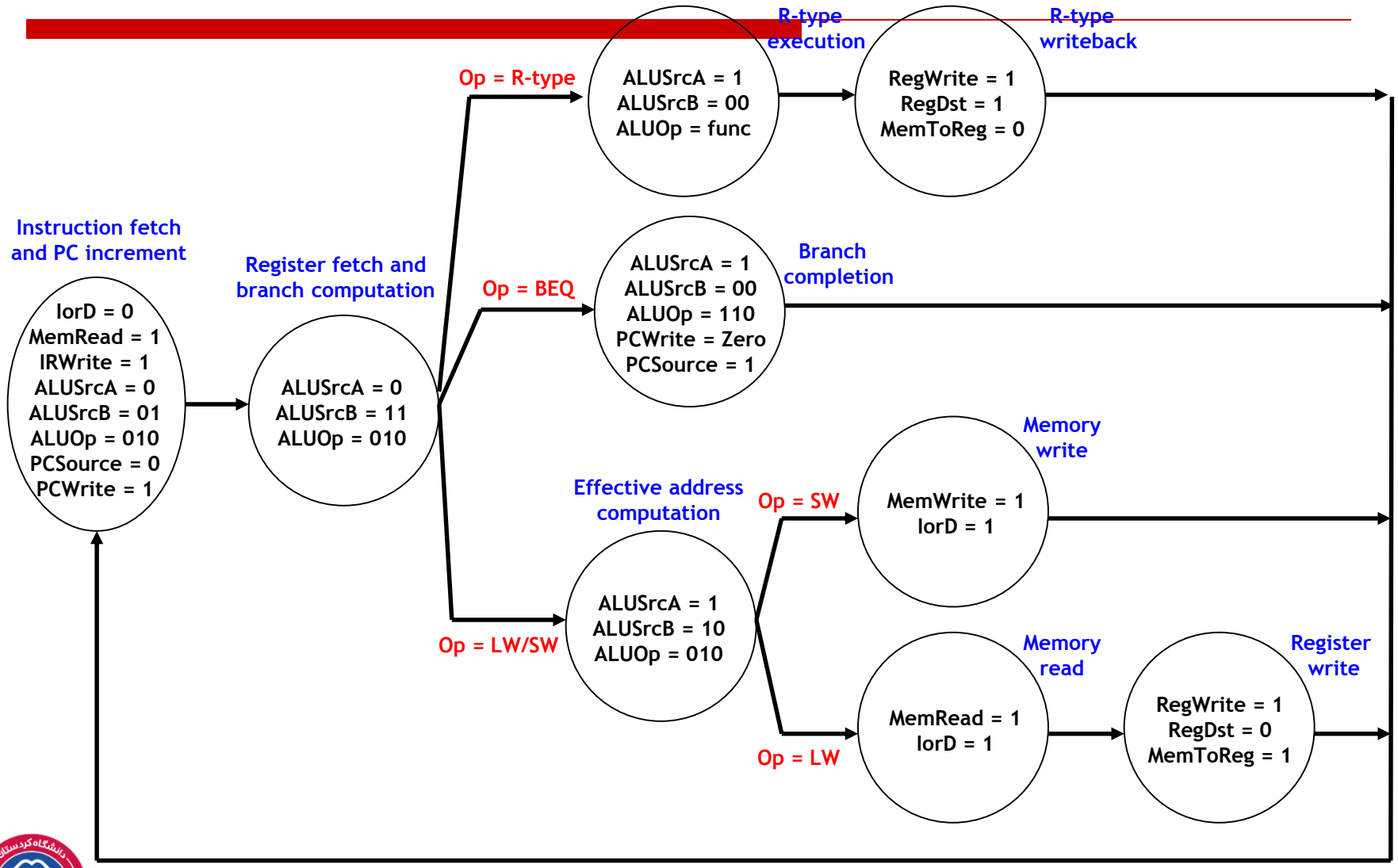
Signal	Value	Description
ALUSrcA	0	Use PC as the first ALU operand
ALUSrcB	01	Use constant 4 as the second ALU operand
ALUOp	ADD	Perform addition
PCWrite	1	Change PC
PCSource	0	Update PC from the ALU output

- We'll assume that all control signals not listed are implicitly set to 0.

Summary of Instruction Execution

Step	Step name	Action for R-type instructions	Action for memory-reference instructions	Action for branches	Action for jumps
1: IF	Instruction fetch	IR = Memory[PC] PC = PC + 4			
2: ID	Instruction decode/register fetch	A = Reg [IR[25-21]] B = Reg [IR[20-16]] ALUOut = PC + (sign-extend (IR[15-0]) << 2)			
3: EX	Execution, address computation, branch/ jump completion	ALUOut = A op B	ALUOut = A + sign-extend (IR[15-0])	if (A ==B) then PC = ALUOut	PC = PC [31-28] II (IR[25-0]<<2)
4: MEM	Memory access or R-type completion	Reg [IR[15-11]] = ALUOut	Load: MDR = Memory[ALUOut] or Store: Memory [ALUOut] = B		
5: WB	Memory read completion		Load: Reg[IR[20-16]] = MDR		

Finite-state machine for the control unit



Implementing the FSM

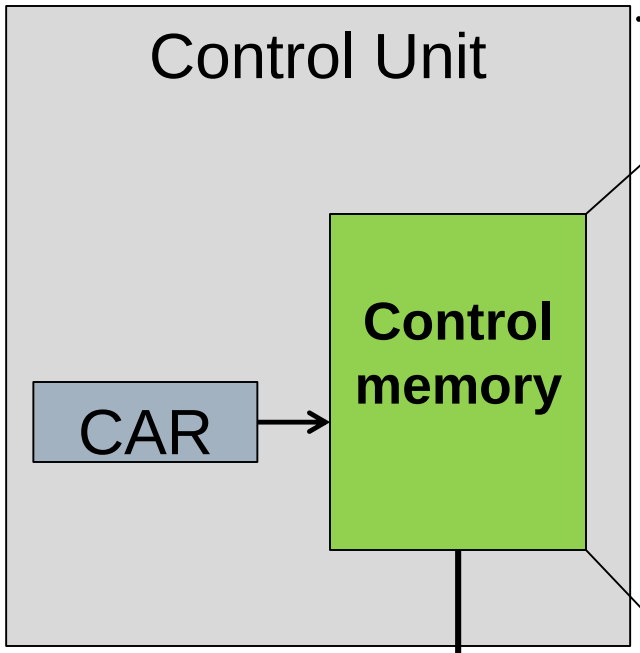
- This can be translated into a state table; here are the first two states.

Current State	Input (Op)	Next State	Output (Control signals)											
			PC Write	IorD	MemRead	MemWrite	IRWrite	RegDst	MemToReg	RegWrite	ALUSrcA	ALUSrcB	ALUOp	PCSource
Instr Fetch	X	Reg Fetch	1	0	1	0	1	X	X	0	0	01	010	0
Reg Fetch	BEQ	Branch compl	0	X	0	0	0	X	X	0	0	11	010	X
Reg Fetch	R-type	R-type execute	0	X	0	0	0	X	X	0	0	11	010	X
Reg Fetch	LW/SW	Compute eff addr	0	X	0	0	0	X	X	0	0	11	010	X

- You can implement this the hard way (hardwired control).
  - Represent the current state using flip-flops or a register.
  - Find equations for the next state and (control signal) outputs in terms of the current state and input (instruction word).
- Or you can use the easy way.
  - Write the whole control signals into a memory, like a ROM.
  - This would be much easier, since you don't have to derive equations.

Control Unit (micro-program)

در کنترل به صورت Micro program اطلاعات کنترلی در حافظه ای موسوم به حافظه کنترلی ذخیره میگردد.

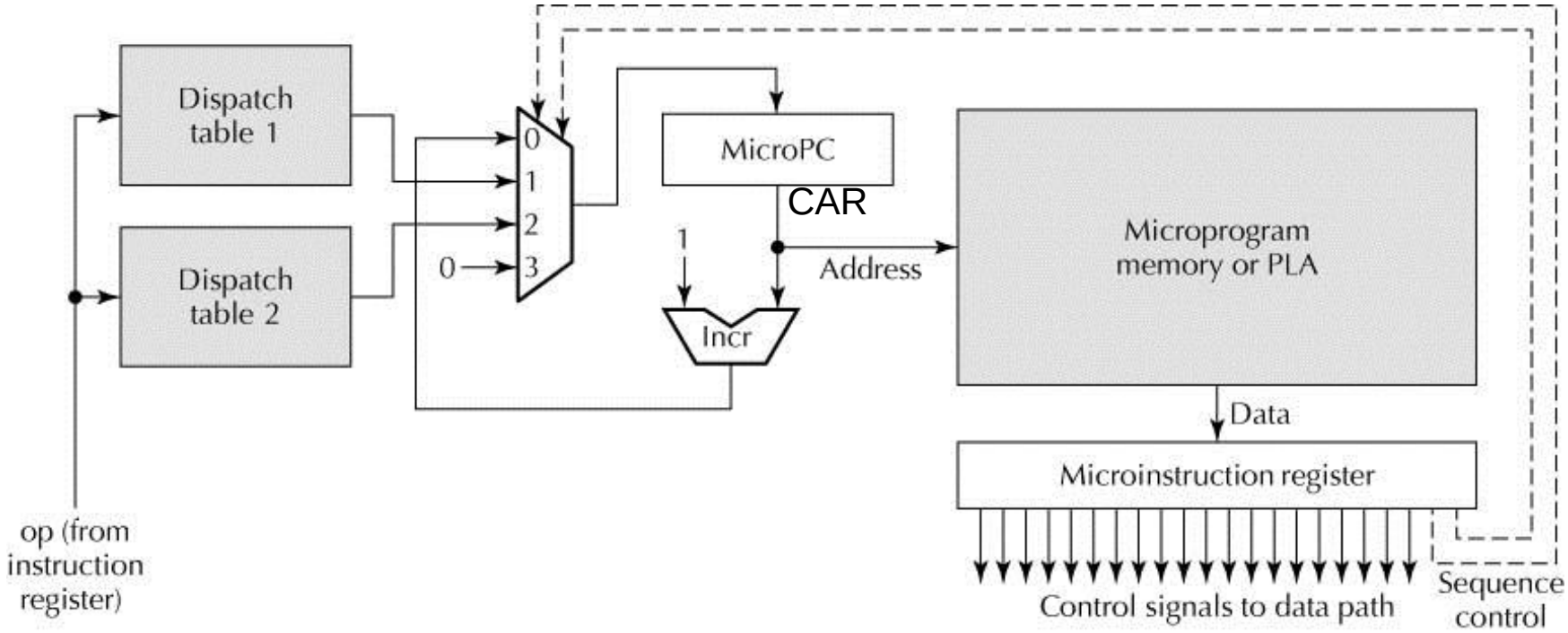


CAR: Control Address Register

Control word To DataPath

Microstore Address	A	AMUX	B	BMUX	C	CMUX	URW	ALU	COND	JUMP ADDR
0	10000	0	10000	0	10010	1	010	0101	000	0000000000000
1	00000	0	00000	0	00000	0	000	0101	111	0000000000000
1152	10010	0	00000	0	00000	1	001	0101	110	1111111111111
1280	10000	0	10000	0	00111	0	000	0101	000	0000000000000
1281	10010	0	10010	0	10000	1	000	1000	000	0000000000000
1282	10000	1	10000	1	10000	1	000	1000	000	0000000000000
1283	10000	0	10000	1	10000	0	000	1000	110	0000000000000
1600	00000	0	00000	0	00000	0	000	0101	101	1100100000010
1601	00000	1	00000	1	00000	1	000	0111	110	1111111111111
1602	10010	0	00000	0	10000	1	000	1100	000	0000000000000
1603	00000	1	10000	1	00000	1	000	0111	110	1111111111111
1604	00000	0	00000	0	00000	0	000	0101	101	1100100001110
1605	00000	1	00000	1	00000	1	000	0000	110	1111111111111
1606	10010	0	00000	0	10000	1	000	1011	000	0000000000000
1607	00000	1	10000	1	00000	1	000	0000	110	1111111111111
1608	00000	0	00000	0	00000	0	000	0101	101	1100100010100
1609	00000	1	00000	1	00000	1	000	0001	110	1111111111111
1610	10010	0	00000	0	10000	1	000	1011	000	0000000000000
1611	00000	1	10000	1	00000	1	000	0001	110	1111111111111
1624	00000	0	00000	0	00000	0	000	0101	101	1100101101010
1625	00000	1	00000	1	00000	1	000	0010	110	1111111111111
1626	10010	0	00000	0	10000	1	000	1011	000	0000000000000
1627	00000	1	10000	1	00000	1	000	0010	110	1111111111111
1688	00000	0	00000	0	00000	0	000	0101	000	1101001101010
1689	00000	1	00000	1	00000	1	000	0100	110	1111111111111
1690	10010	0	00000	0	10000	1	000	1011	000	0000000000000
1691	00000	1	10000	1	00000	1	000	0100	110	1111111111111
1760	00000	0	00000	0	00000	0	000	0101	101	1101110001010
1761	00000	1	00000	1	10000	0	000	1000	110	0000000000000
1762	10010	0	00000	0	10000	1	000	1100	000	0000000000000
1763	00000	1	10000	1	00000	0	000	1000	110	0000000000000
1792	00000	1	00000	1	10000	1	000	1000	101	1110000000010

Control Unit (micro-program)



Control Unit (micro-program)

Microprogram containing 10 microinstructions

Label	ALU control	SRC1	SRC2	Register control	Memory	PCWrite control	Sequencing
Fetch	Add	PC	4		Read PC	ALU	Seq
	Add	PC	Extshft	Read			Dispatch 1
Mem1	Add	A	Extend				Dispatch 2
LW2					Read ALU		Seq
				Write MDR			Fetch
SW2					Write ALU		Fetch
Rformat1	Func code	A	B				Seq
				Write ALU			Fetch
BEQ1	Subt	A	B			ALUOut-cond	Fetch
JUMP1						Jump address	Fetch

Dispatch Table 1

Dispatch ROM 1		
Op	Opcode name	Value
000000	R-format	Rformat1
000010	jmp	JUMP1
000100	beq	BEQ1
100011	lw	Mem1
101011	sw	Mem1

Dispatch Table 2

Dispatch ROM 2		
Op	Opcode name	Value
100011	lw	LW2
101011	sw	SW2

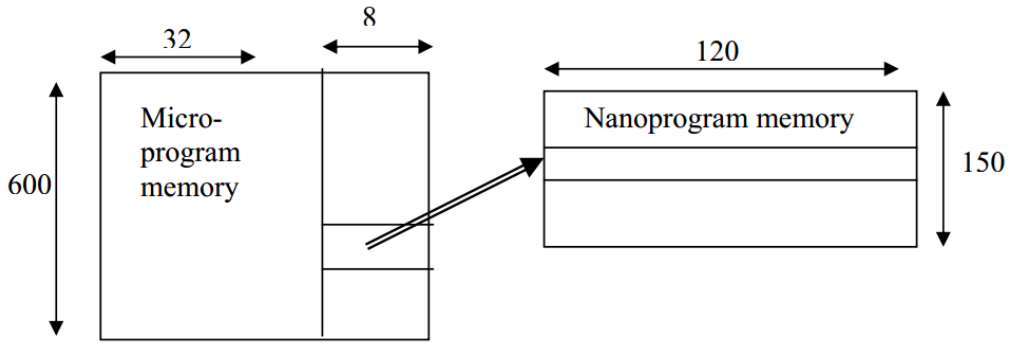
حافظه کنترلی (کنترل به روش ریزبرنامه)

1600	0 0 0 0 0 0	0	0 0 0 0 0 0	0	0 0 0 0 0 0	0	0	0	0 1 0 1	1 0 1	1 1 0 0 1 0 0 0 0 1 0
1601	0 0 0 0 0 0	1	0 0 0 0 0 0	1	0 0 0 0 0 0	1	0	0	0 0 1 1	1 1 0	1 1 1 1 1 1 1 1 1 1 1
1602	1 0 0 1 0 1	0	0 0 0 0 0 0	0	1 0 0 0 0 1	0	0	0	1 1 0 0	0 0 0	0 0 0 0 0 0 0 0 0 0 0
1603	0 0 0 0 0 0	1	1 0 0 0 0 1	0	0 0 0 0 0 0	1	0	0	0 0 1 1	1 1 0	1 1 1 1 1 1 1 1 1 1 1
1604	0 0 0 0 0 0	0	0 0 0 0 0 0	0	0 0 0 0 0 0	0	0	0	0 1 0 1	1 0 1	1 1 0 0 1 0 0 0 1 1 0
1605	0 0 0 0 0 0	1	0 0 0 0 0 0	1	0 0 0 0 0 0	1	0	0	0 0 0 0	1 1 0	1 1 1 1 1 1 1 1 1 1 1
1606	1 0 0 1 0 1	0	0 0 0 0 0 0	0	1 0 0 0 0 1	0	0	0	1 0 1 1	0 0 0	0 0 0 0 0 0 0 0 0 0 0
1607	0 0 0 0 0 0	1	1 0 0 0 0 1	0	0 0 0 0 0 0	1	0	0	0 0 0 0	1 1 0	1 1 1 1 1 1 1 1 1 1 1
1608	0 0 0 0 0 0	0	0 0 0 0 0 0	0	0 0 0 0 0 0	0	0	0	0 1 0 1	1 0 1	1 1 0 0 1 0 0 1 0 1 0
1609	0 0 0 0 0 0	1	0 0 0 0 0 0	1	0 0 0 0 0 0	1	0	0	0 0 0 1	1 1 0	1 1 1 1 1 1 1 1 1 1 1
1610	1 0 0 1 0 1	0	0 0 0 0 0 0	0	1 0 0 0 0 1	0	0	0	1 0 1 1	0 0 0	0 0 0 0 0 0 0 0 0 0 0
1611	0 0 0 0 0 0	1	1 0 0 0 0 1	0	0 0 0 0 0 0	1	0	0	0 0 0 1	1 1 0	1 1 1 1 1 1 1 1 1 1 1
1624	0 0 0 0 0 0	0	0 0 0 0 0 0	0	0 0 0 0 0 0	0	0	0	0 1 0 1	1 0 1	1 1 0 0 1 0 1 1 0 1 0
1625	0 0 0 0 0 0	1	0 0 0 0 0 0	1	0 0 0 0 0 0	1	0	0	0 0 1 0	1 1 0	1 1 1 1 1 1 1 1 1 1 1

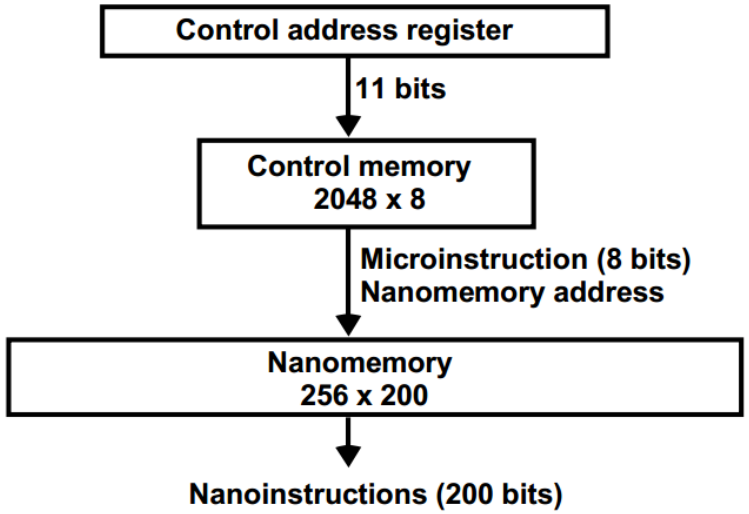
در کامپیوترهای پیچیده تر حافظه کنترلی ممکن است شامل خانه های تکراری باشد

حافظه کنترلی دو سطحی (استفاده از نانو حافظه)

اگر در یک سیستم نیاز باشد که تعداد بسیار زیادی سیگنال کنترلی تولید شود و نیز تعداد سیگنالهای کنترلی مجزا محدود و تعداد کمی باشند، به جای قرار دادن تمامی این سیگنالها در حافظه میکرو، فقط سیگنالهای مستقل را در یک حافظه تحت عنوان حافظه نانو قرار داده و در حافظه میکرو آدرس این سیگنالها(کلمات) کنترلی را مشخص میکنیم. در این حالت، ما دیگر تعداد زیاد و تکراری کلمات کنترلی را در حافظه میکرو نداریم و به جای آن آدرسهای تکراری که حافظه کمتری اشغال میکنند را داریم.



مثال ۲



مثال ۱

حافظه کنترلی دو سطحی (استفاده از نانو حافظه)

مثال: فرض کنید که در یک سیستم تعداد ۳۰۰ کلمه کنترلی را داریم. از این ۳۰۰ تا تعداد ۶۰ کلمه مستقل هستند(۲۴۰ تا تکراری هستند). اگر طول کلمات کنترلی ۱۵۰ بیت باشد(تعداد سیگنالهای کنترلی ۱۵۰ باشد) خواهیم داشت:

۱- در حالت معمول و بدون استفاده از حافظه نانو، حجم حافظه میکرو برابر ۳۰۰*۱۵۰ است.

۲- در صورت استفاده از حافظه نانو ، حجم این حافظه برابر ۶۰*۱۵۰ است. زیرا در این حافظه فقط قرار است که کلمات کنترلی مستقل و غیر تکراری قرار گیرند.

در حالت دوم، حافظه میکرو فقط باید آدرس ۳۰۰ تا کلمه کنترلی مورد نیاز را از حافظه نانو مشخص نماید. برای مشخص کردن (آدرس دهی) ۶۰ کلمه کنترلی، به ۶ بیت احتیاج داریم. چون سیستم به ۳۰۰ سیگنال کنترلی نیاز دارد، پس در این حالت حجم حافظه میکرو برابر است با: ۳۰۰*۶.

میزان صرفه جویی: ۳۴۲۰۰bits = (۳۰۰*۶ + ۶۰*۱۵۰)-(۳۰۰*۱۵۰)

حافظه کنترلی دو سطحی (استفاده از نانو حافظه)

یک پردازنده دارای ۱۷۵ سیگنال کنترلی و ۲۵۰ ریزدستور است. اگر۲۰۰ ریزدستور متفاوت در این پردازنده وجود داشته باشد، حجم کل حافظه واحد کنترل در صورت استفاده از حافظه نانو چقدر است؟

الف- ۲۸۰۰۰بیت
ب- ۳۲۰۰۰بیت
ج- ۲۴۰۰۰بیت
د- ۳۷۰۰۰بیت

Summary

- A single-cycle CPU has two main disadvantages.
  - The cycle time is limited by the worst case latency.
  - It requires more hardware than necessary.
- A multicycle processor splits instruction execution into several stages.
  - Instructions only execute as many stages as required.
  - Each stage is relatively simple, so the clock cycle time is reduced.
  - Functional units can be reused on different cycles.
- We made several modifications to the single-cycle datapath.
  - The two extra adders and one memory were removed.
  - Multiplexers were inserted so the ALU and memory can be used for different purposes in different execution stages.
  - New registers are needed to store intermediate results.


Questions