Processor Design

- برای طراحی پردازنده سه *دسته دستور* را انتخاب می‌کنیم:

1- Arithmetic Logical Instructions (R-Type): *add, sub, and, or, slt*
2- Memory Reference Instructions: *lw, sw*
3- Control Flow Instructions: *beq, j*

- برای تمام دستورات دو مرحله‌ی اول یکسان است:

۱- PC به حافظه‌ی دستور ارسال می‌شود و دستور از حافظه واکشی می‌شود

۲- با استفاده از فیلدهای دستور، یک یا دو رجیستر خوانده می‌شود (برای دستور lw فقط یک رجیستر و برای سایر دستورات دو رجیستر خوانده می‌شود)

۳- پس از خواندن رجیسترها، همه‌ی دستورات از ALU استفاده می‌کنند

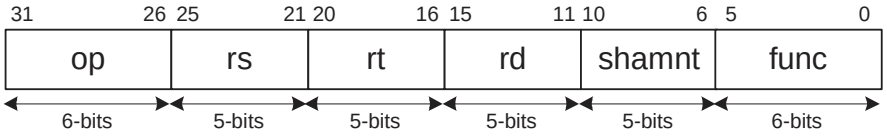
- دستورات دسته‌ی اول (Memory Reference) از ALU برای *محاسبه‌ی آدرس* استفاده می‌کنند
- دستورات دسته‌ی دوم (Arithmetic Logical) از ALU برای *محاسبه‌ی یک عملیات محاسباتی-منطقی* استفاده می‌کنند
- دستورات دسته‌ی سوم (Control Flow) از ALU برای *مقایسه* استفاده می‌کنند

٤- از این مرحله دستورات به صورت متفاوت عمل می‌کنند

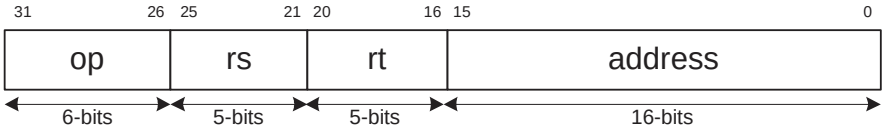
- دستورات دسته‌ی اول به حافظه دسترسی پیدا می‌کنند (برای خواندن یا نوشتن)
- دستورات دسته‌ی دوم حاصل ALU را در رجیستر فایل ذخیره می‌کنند
- تغییر احتمالی PC بر اساس نتیجه‌ی مقایسه

- قالب دستورات انتخابی

add, sub, and, or, slt



lw, sw, beq



- اجزا مورد نیاز برای طراحی مسیر داده

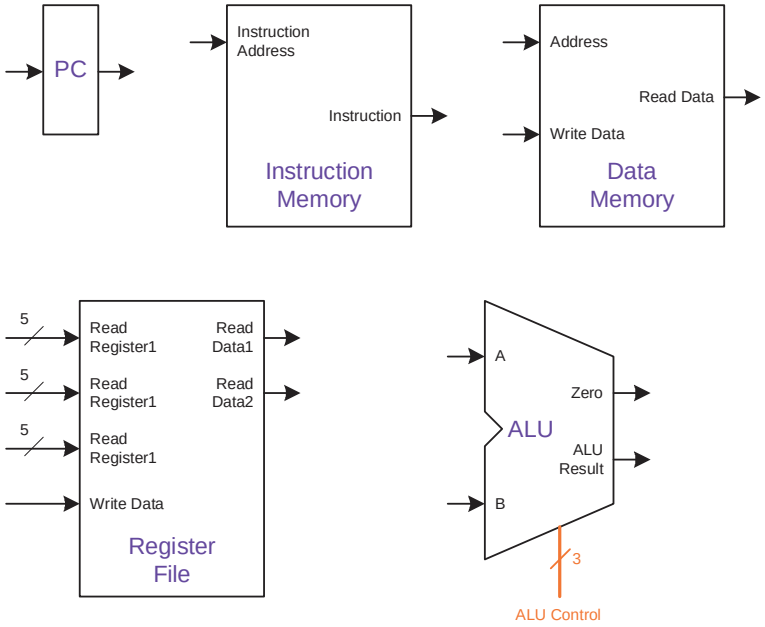
۱- Program Counter

۲- حافظه‌ی دستور Instruction Memory

۳- حافظه‌ی داده Data Memory

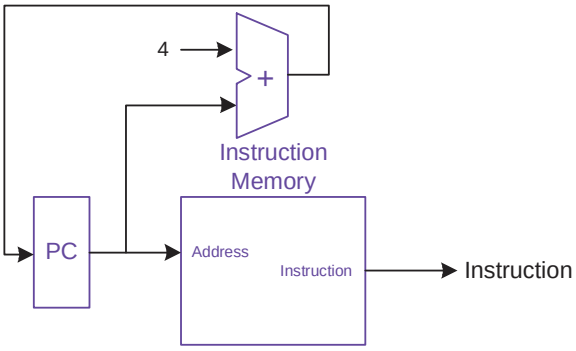
٤- رجیستر فایل Register File

۵- واحد محاسباتی-منطقی ALU

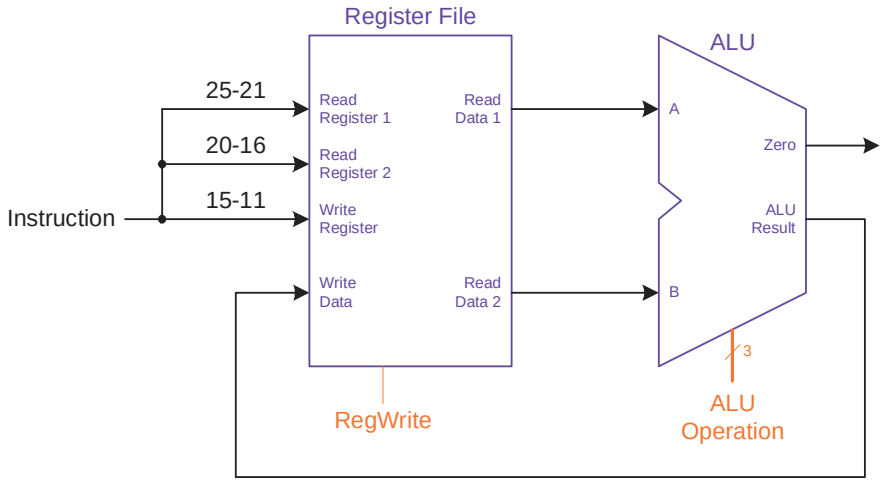


- *طراحی مسیر داده*

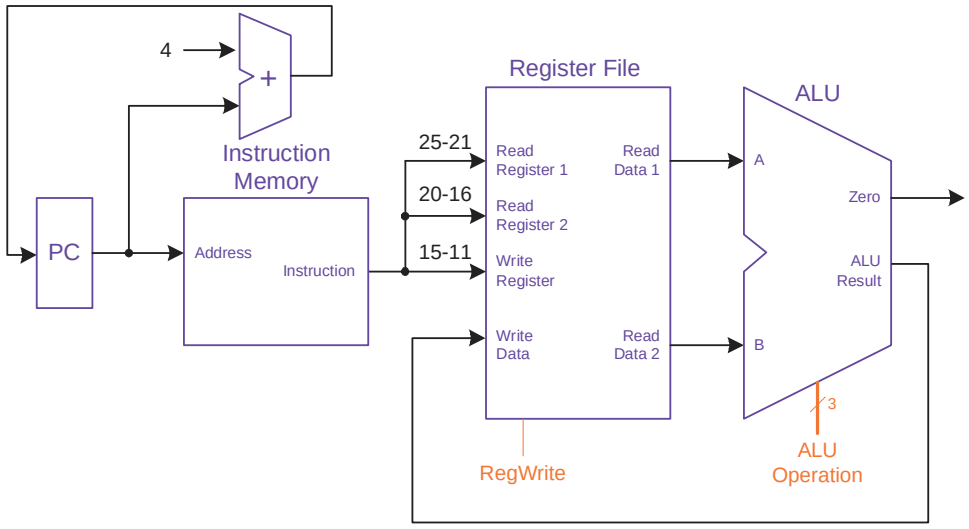
مرحله‌ی اول: طراحی مسیر داده‌ی بخش Program Sequencing



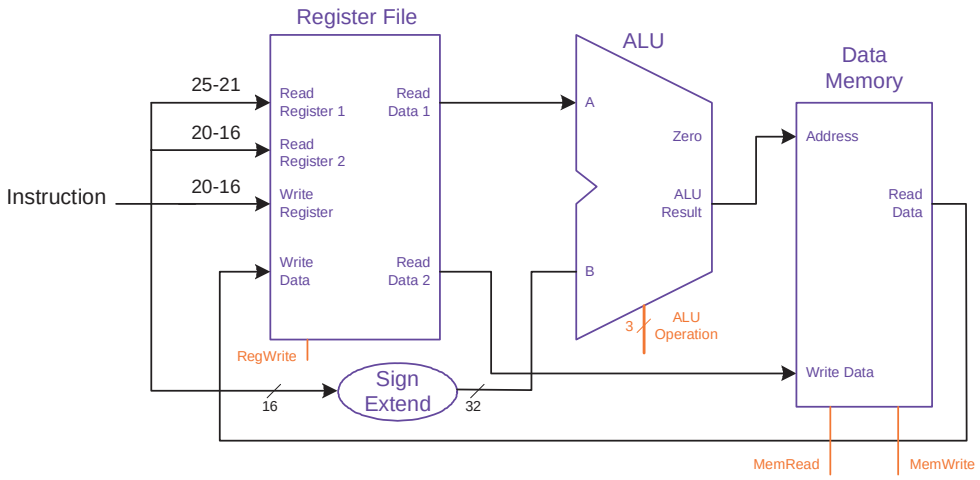
مرحله‌ی دوم: طراحی مسیر داده برای دستورات R-Type



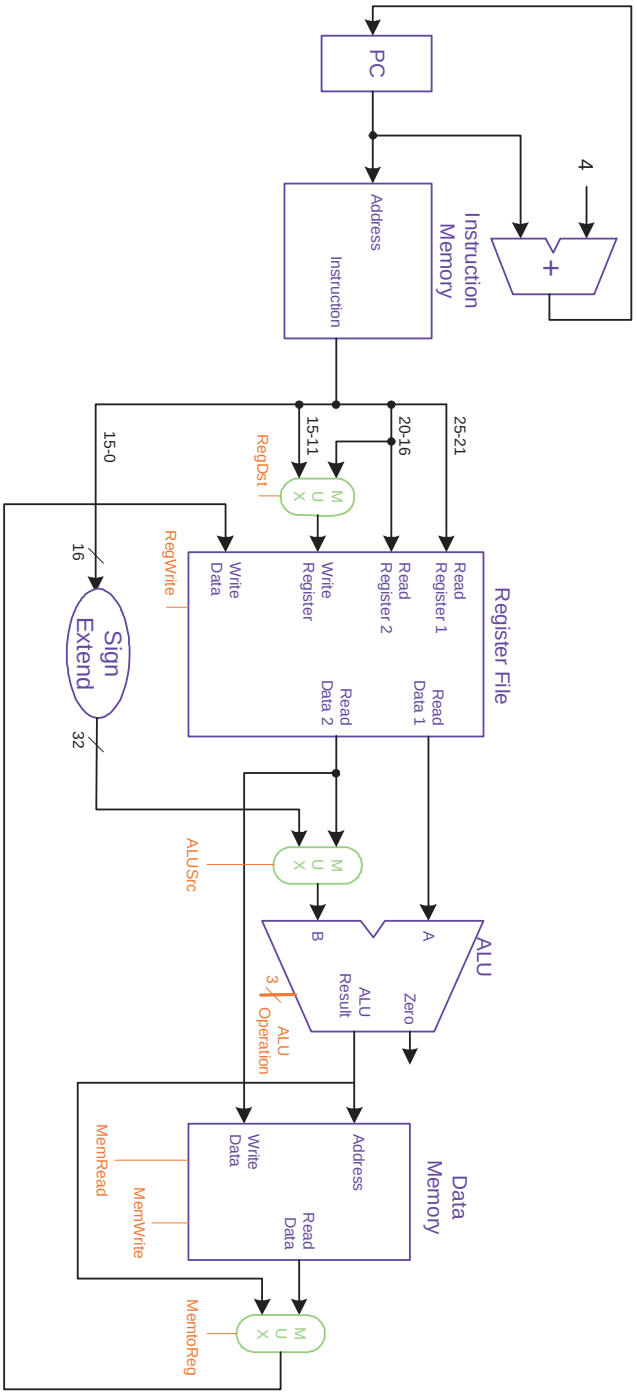
مرحله‌ی سوم: ادغام دو بخش اول و دوم



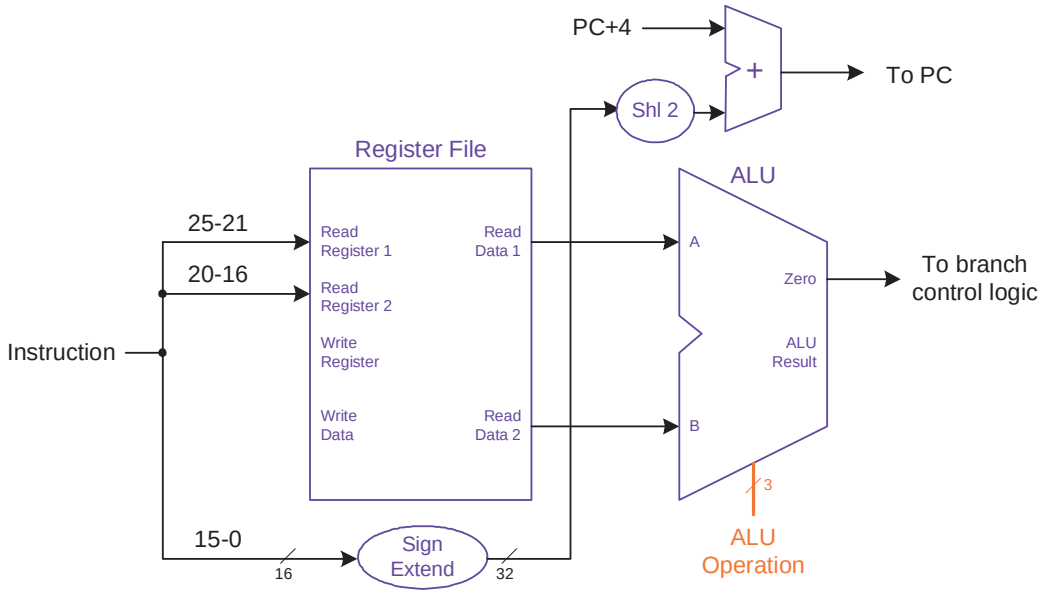
مرحله‌ی چهارم: طراحی مسیر داده برای دستورات lw, sw



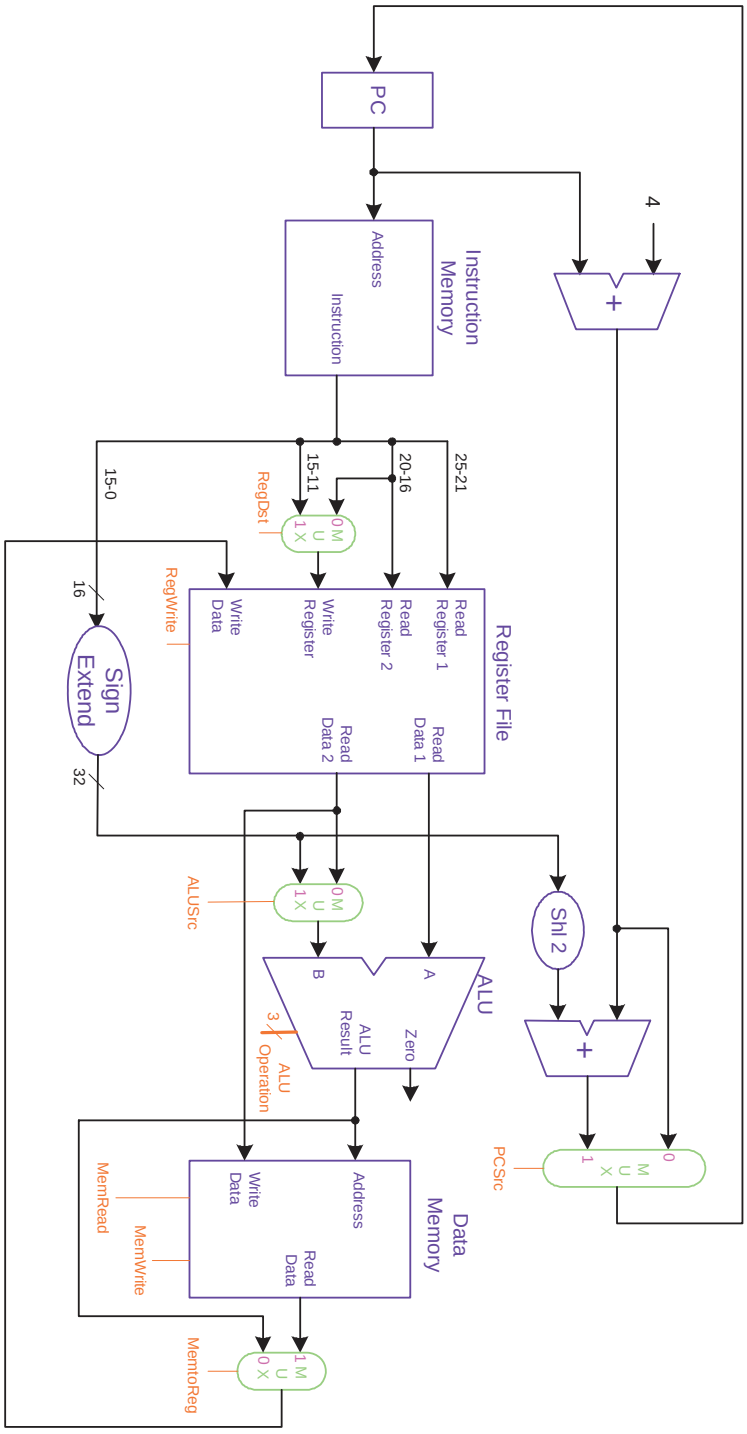
مرحله‌ی پنجم: ادغام دو بخش سوم و چهارم



مرحله‌ی ششم: طراحی مسیر داده برای دستور beq



مرحله‌ی هفتم: ادغام دو بخش پنجم و ششم



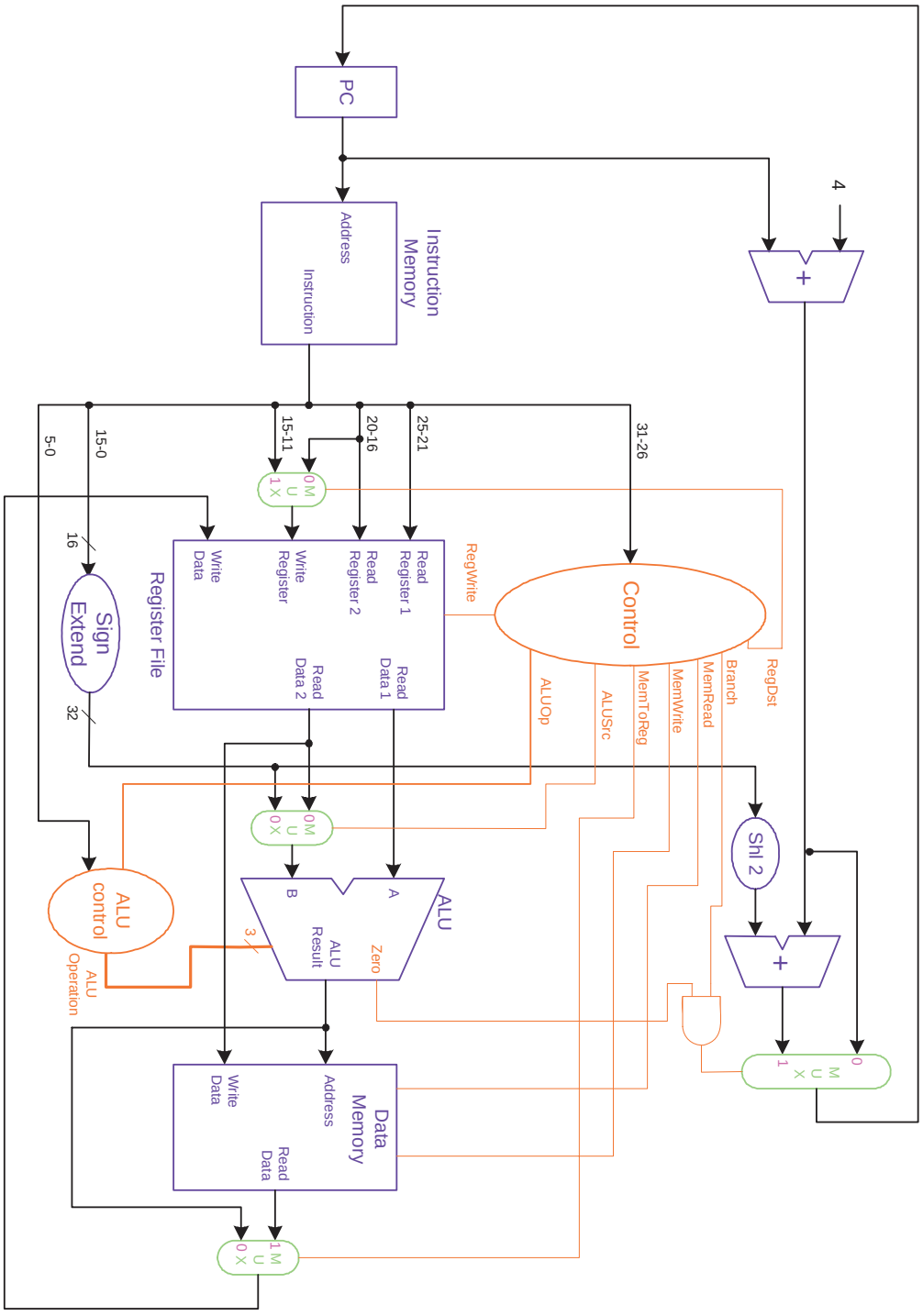
- *طراحی واحد کنترل*
  - ALU Control

ALU Control Input	Function
000	AND
001	OR
010	Add
110	Sub
111	Set on less than

- در دستورات *lw* و *sw* از ALU برای *محاسبه‌ی آدرس* استفاده می‌شود (جمع)
- در دستور *beq* از ALU برای تست برابری دو رجیستر استفاده می‌شود (*تفریق*)
- در دستورات *R-Type* از ALU برای انجام *عملیات محاسباتی* استفاده می‌شود (*براساس بیتهای Function*)

Instruction opcode	ALUOp	Instruction operation	Func. field	Desired ALU action	ALU control input
lw	00	Load word	XXXXXX	Add	010
sw	00	Store word	XXXXXX	Add	010
beq	01	Branch equal	XXXXXX	Sub	110
R-Type	10	Add	100000	Add	010
R-Type	10	Sub	100010	Sub	110
R-Type	10	And	100100	And	000
R-Type	10	Or	100101	Or	001
R-Type	10	Set on less than	101010	Set on less than	111

ALUOp		Function field						Operation
1	0	F5	F4	F3	F2	F1	F0	
0	0	X	X	X	X	X	X	010
X	1	X	X	X	X	X	X	110
1	X	X	X	0	0	0	0	010
1	X	X	X	0	0	1	0	110
1	X	X	X	0	1	0	0	000
1	X	X	X	0	1	0	1	001
1	X	X	X	1	0	1	0	111


PC
Address
Instruction
Instruction Memory
4
+
5-0
15-0
15-11
20-16
25-21
31-26
M U X
0
1
16
Sign Extend
32
Write Data
Write Register
Read Register 2
Read Register 1
Read Data 2
Read Data 1
Register File
RegWrite
Control
RegDst
Branch
MemRead
MemWrite
MemToReg
ALUSrc
ALUOp
ALU control
3
ALU Operation
B
A
ALU
ALU Result
Zero
Shl 2
Write Data
Address
Read Data
Data Memory

- جدول صحت واحد کنترل

Instruction	Reg Dst	ALU Src	Mem To Reg	Reg Write	Mem Read	Mem Write	Branch	ALU Op1	ALU Op0
R-Type	1	0	0	1	0	0	0	1	0
Lw	0	1	1	1	1	0	0	0	0
Sw	X	1	X	0	0	1	0	0	0
Beq	X	0	X	0	0	0	1	0	1

- به علت اینکه تمام مراحل اجرای دستور در یک سیکل Clock انجام می‌شود، به این پیاده‌سازی، پیاده‌سازی *تک مرحله‌ای Single Cycle* گفته می‌شود.

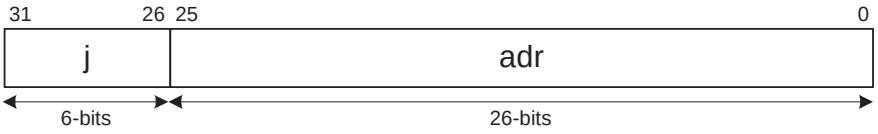
- نحوه‌ی اجرای *دستورات R-Type* (مثال: add $1, $2, $3)
  - دستور از حافظه‌ی دستور، واکشی (Fetch) می‌شود و PC افزایش می‌یابد.
  - دو رجیستر ($2, $3) از رجیستر فایل خوانده می‌شود.
  - ALU روی داده‌های خوانده شده از رجیسترفایل عمل می‌کند (براساس بیت‌های ۰ تا ۵ دستور) و خروجی ALU تولید می‌شود.
  - نتیجه‌ی ALU در رجیستر در $1 رجیسترفایل نوشته می‌شود.

- نحوه‌ی اجرای *دستور lw* (مثال: lw $1, offset($2))
  - دستور از حافظه‌ی دستور، واکشی (Fetch) می‌شود و PC افزایش می‌یابد.
  - رجیستر $2 از رجیستر فایل خوانده می‌شود.
  - ALU حاصل جمع مقدار خوانده شده از رجیسترفایل و گسترش علامت یافته‌ی offset را محاسبه می‌کند.
  - خروجی ALU به عنوان آدرس حافظه‌ی داده استفاده شده و داده از حافظه‌ی داده خوانده می‌شود.
  - داده‌ی خوانده شده از حافظه‌ی داده در رجیستر $1 از رجیستر فایل نوشته می‌شود.

- نحوه‌ی اجرای *دستور sw* (مثال: sw $1, offset($2))
  - دستور از حافظه‌ی دستور، واکشی (Fetch) می‌شود و PC افزایش می‌یابد.
  - دو رجیستر ($1, $2) از رجیستر فایل خوانده می‌شود.
  - ALU حاصل جمع مقدار خوانده شده از رجیسترفایل و گسترش علامت یافته‌ی offset را محاسبه می‌کند.
  - خروجی ALU به عنوان آدرس حافظه استفاده می‌شود و رجیستر $1 در حافظه‌ی داده نوشته می‌شود.

- نحوه‌ی اجرای دستور *beq* (مثال: beq $1, $2, offset)
  - دستور از حافظه‌ی دستور، واکشی (Fetch) می‌شود و PC افزایش می‌یابد.
  - دو رجیستر ($1, $2) از رجیستر فایل خوانده می‌شود.
  - ALU عملیات تفریق را روی داده‌های خوانده شده از رجیسترفایل انجام می‌دهد. مقدار افزایش یافته‌ی PC (PC+4) با گسترش علامت‌یافته‌ی offset که دو بیت به سمت راست شیفت یافته، جمع می‌شود و حاصل به عنوان آدرس پرش مقصد در نظر گرفته می شود.
  - حاصل Zero از ALU برای تصمیم‌گیری مقدار بعدی PC استفاده می‌شود.

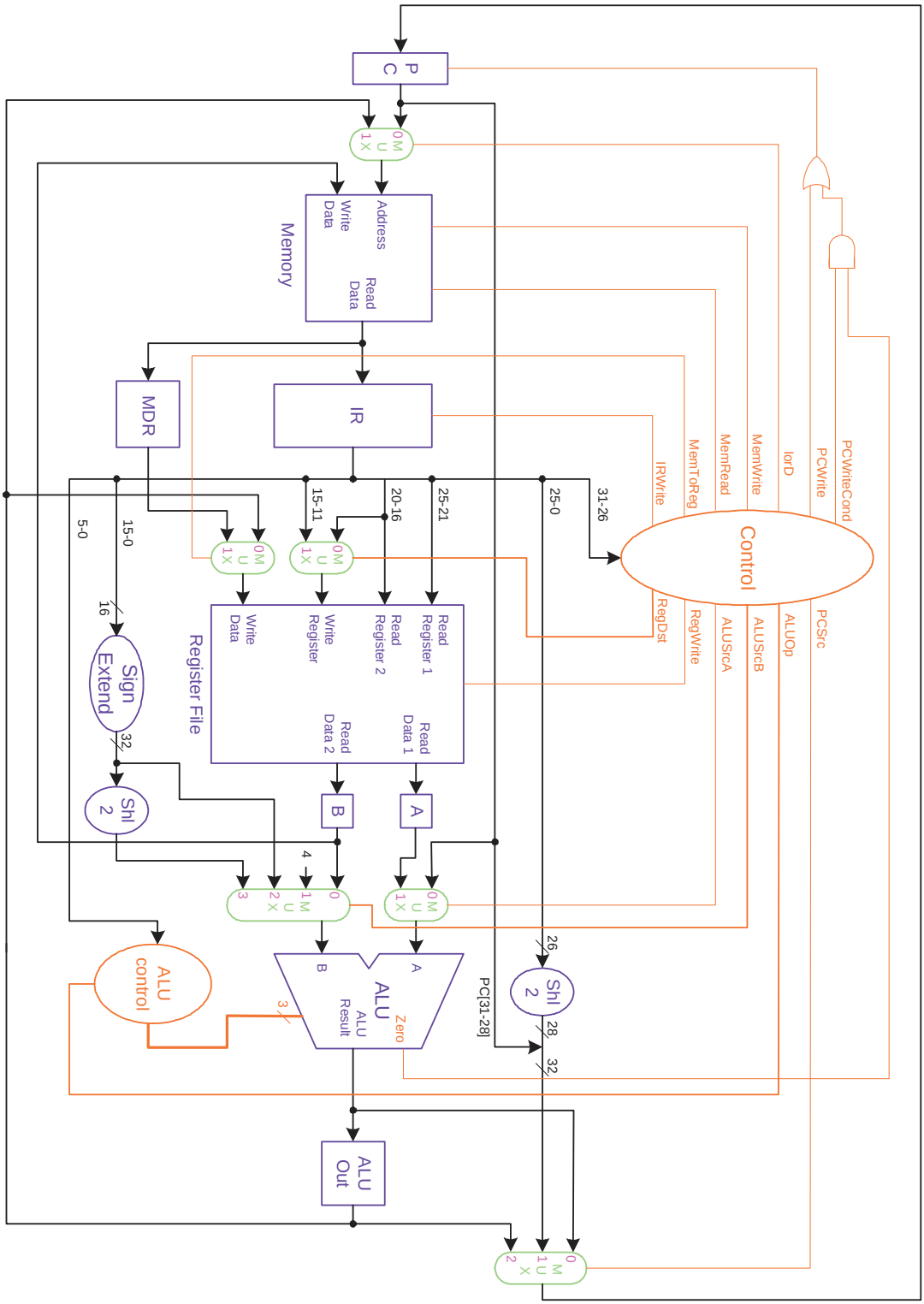
- نحوه‌ی افزودن دستور *j* (مثال: j adr)



  - آدرسی که باید به آن پرش کنیم (داخل رجیستر PC لود شود) از concatenation سه بخش زیر تشکیل می‌شود:
    - ٤ بیت رتبه‌ی بالای PC+4 (بیتهای 28-31)
    - ۲۶ بیت آدرس موجود در دستور (بیتهای 0-25)
    - ۲ بیت 00

- در پیاده‌سازی *تک‌مرحله‌ای*، هر دستور به مجموعه‌ای از مراحل مرتبط با واحدهای عملیاتی مورد نیاز تقسیم می‌شد.
- در پیاده‌سازی چندمرحله‌ای *Multi-Cycle* ، هر دستور به *تعدادی مرحله* تقسیم می‌شود و اجرای هر مرحله یک *Clock Cycle* طول می‌کشد. این امر اجازه می‌دهد که *یک واحد عملیاتی* در یک دستور، *بیش از یک بار* مورد استفاده قرار گیرد. بنابراین به تعداد واحدهای عملیاتی کمتری نیاز است.
- دو *ویژگی مهم پیاده‌سازی چندمرحله‌ای*:
  ۱- *متغیر بودن تعداد* *Clock Cycle* های دستورات
  ۲- قابلیت به *اشتراک گذاشتن واحدهای عملیاتی در طی اجرای یک دستور*

- واحدهای موجود برای *پیاده‌سازی چندمرحله‌ای*:
  ۱- *یک واحد حافظه* برای دستور و داده
  ۲- *یک واحد* *ALU* به جای یک ALU و دو جمع‌کننده
  ۳- *یک یا چند رجیستر* پس از *واحدهای اصلی عملیاتی* اضافه شده‌اند تا خروجی آن‌ها را برای استفاده در سیکل بعدی حفظ کند. این رجیسترها از دید کاربر پنهان هستند.
  IR (*Instruction Register*): برای نگهداری دستور خوانده شده از حافظه
  MDR (*Memory Data Register*): برای نگهداری داده خوانده شده از حافظه
  A, B: برای نگداری مقادیر اپرندهای خوانده شده از رجیستر فایل
  ALUOut: برای نگداری نتیجه‌ی حاصل از ALU
  نکته‌ی مهم: تمام رجیسترها به جز *IR* مقادیر داده را در *Clock* های متوالی نگهداری می‌کنند، بنابراین به سیگنال کنترلی نیاز ندارند.


PC
M U X
Memory
Address
Write Data
Read Data
MDR
IR
IRWrite
MemToReg
MemRead
MemWrite
IorD
PCWrite
PCWriteCond
Control
RegDst
RegWrite
ALUSrcA
ALUSrcB
ALUOp
PCSrc
25-0
31-26
15-11
20-16
25-21
5-0
15-0
16
Sign Extend
32
Shl 2
Register File
Write Data
Write Register
Read Register 1
Read Register 2
Read Data 1
Read Data 2
B
A
4
ALU control
ALU
Zero
ALU Result
3
26
28
32
PC[31-28]
ALU Out