

- اجزا مورد نیاز برای طراحی مسیر داده

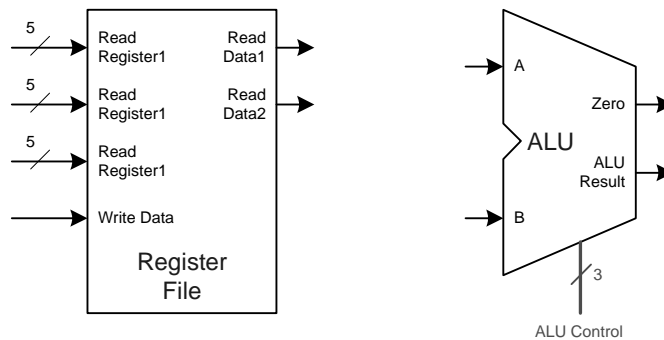
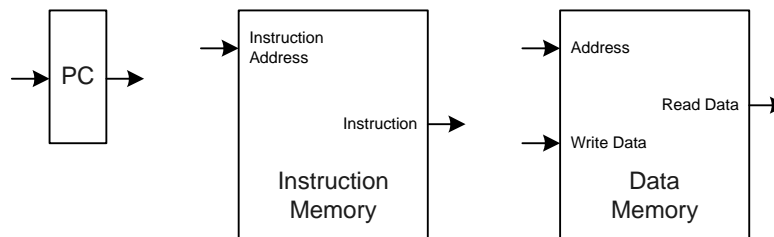
۱- Program Counter

۲- حافظه‌ی دستور Instruction Memory

۳- حافظه‌ی داده Data Memory

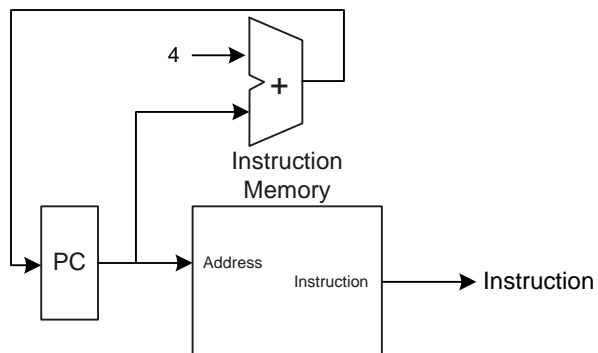
۴- رجیستر فایل Register File

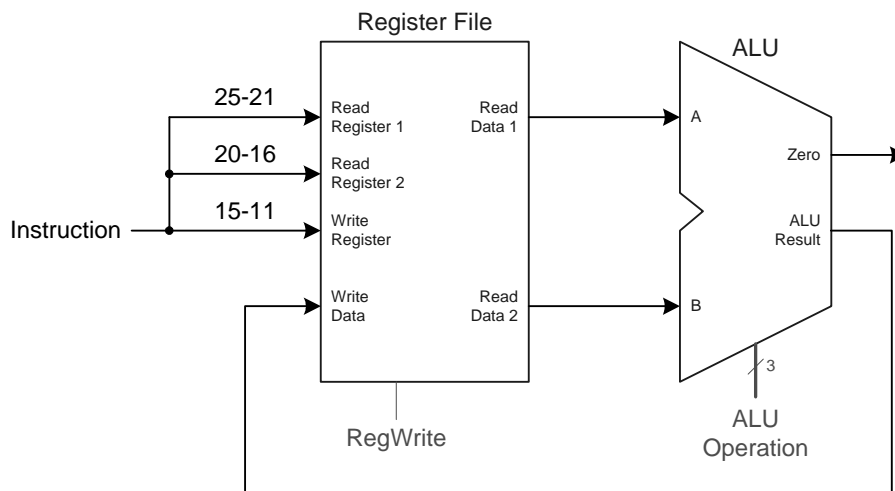
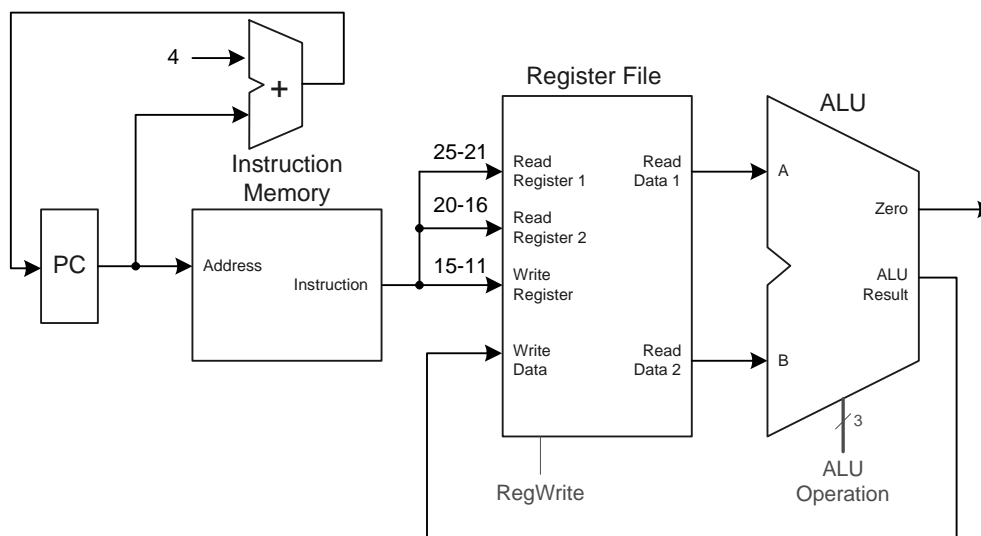
۵- واحد محاسباتی-منطقی ALU



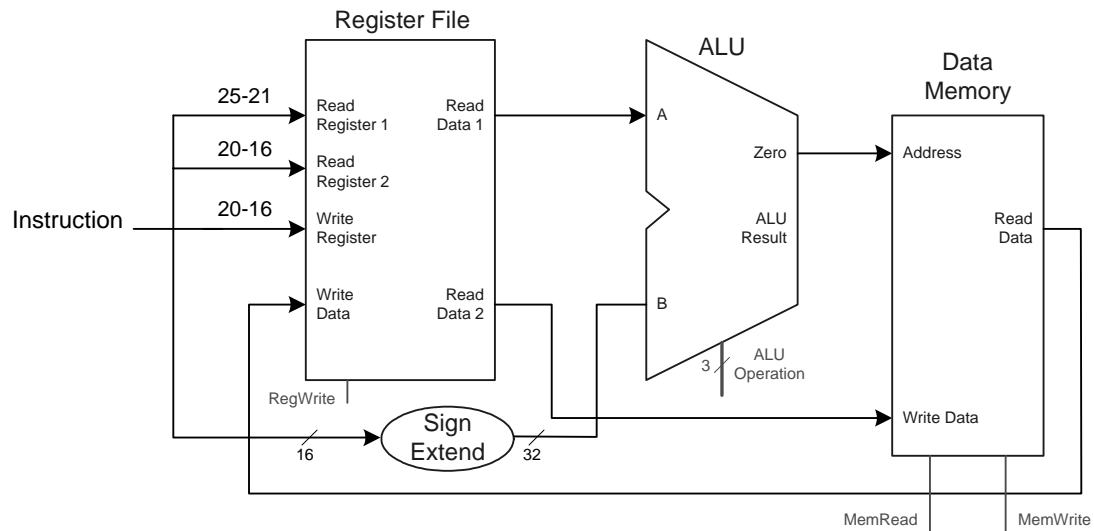
- طراحی مسیر داده

مرحله‌ی اول: طراحی مسیر داده‌ی بخش Program Sequencing

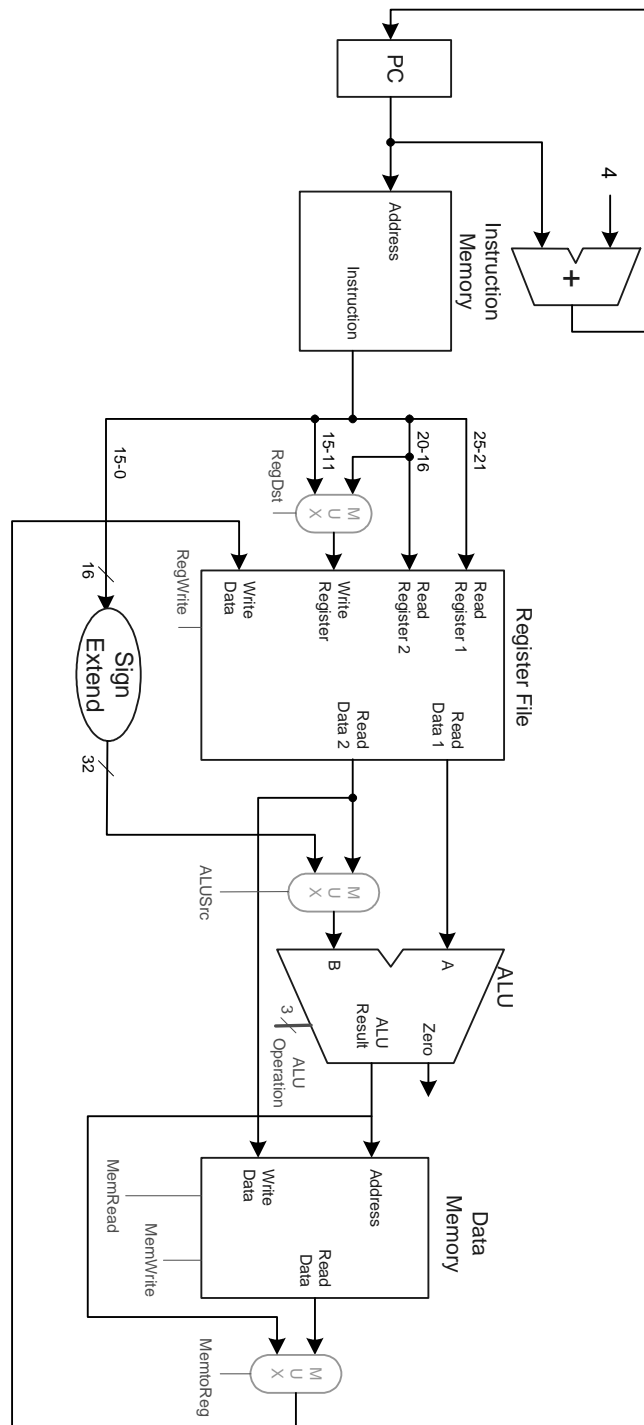


مرحله ی دوم: طراحی مسیر داده برای دستورات R-Typeمرحله ی سوم: ادغام دو بخش اول و دوم

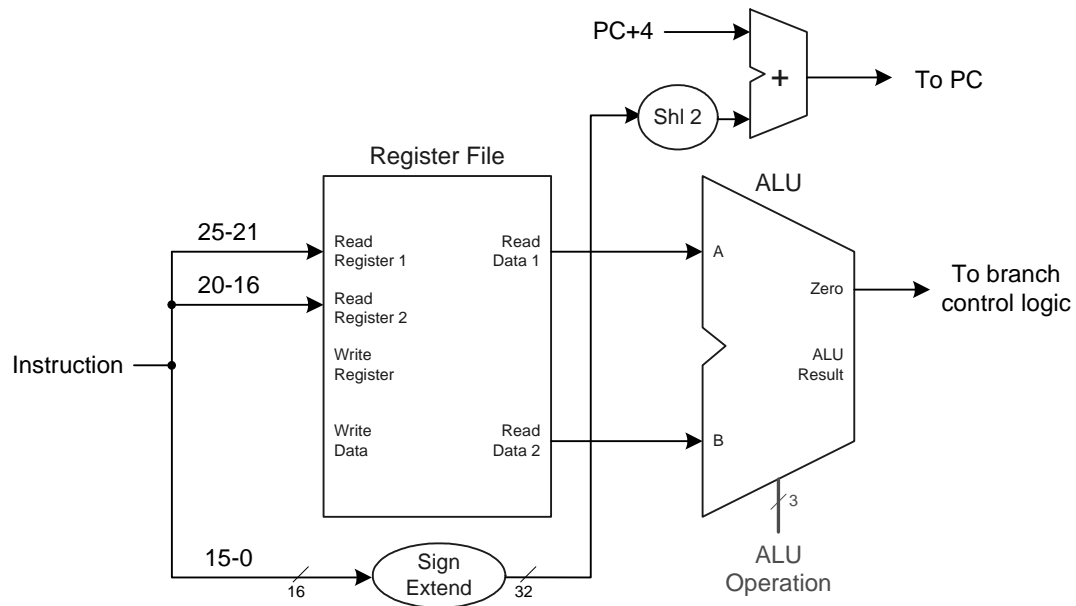
*مرحله ی چهارم: طراحی مسیر داده برای دستورات lw, sw*



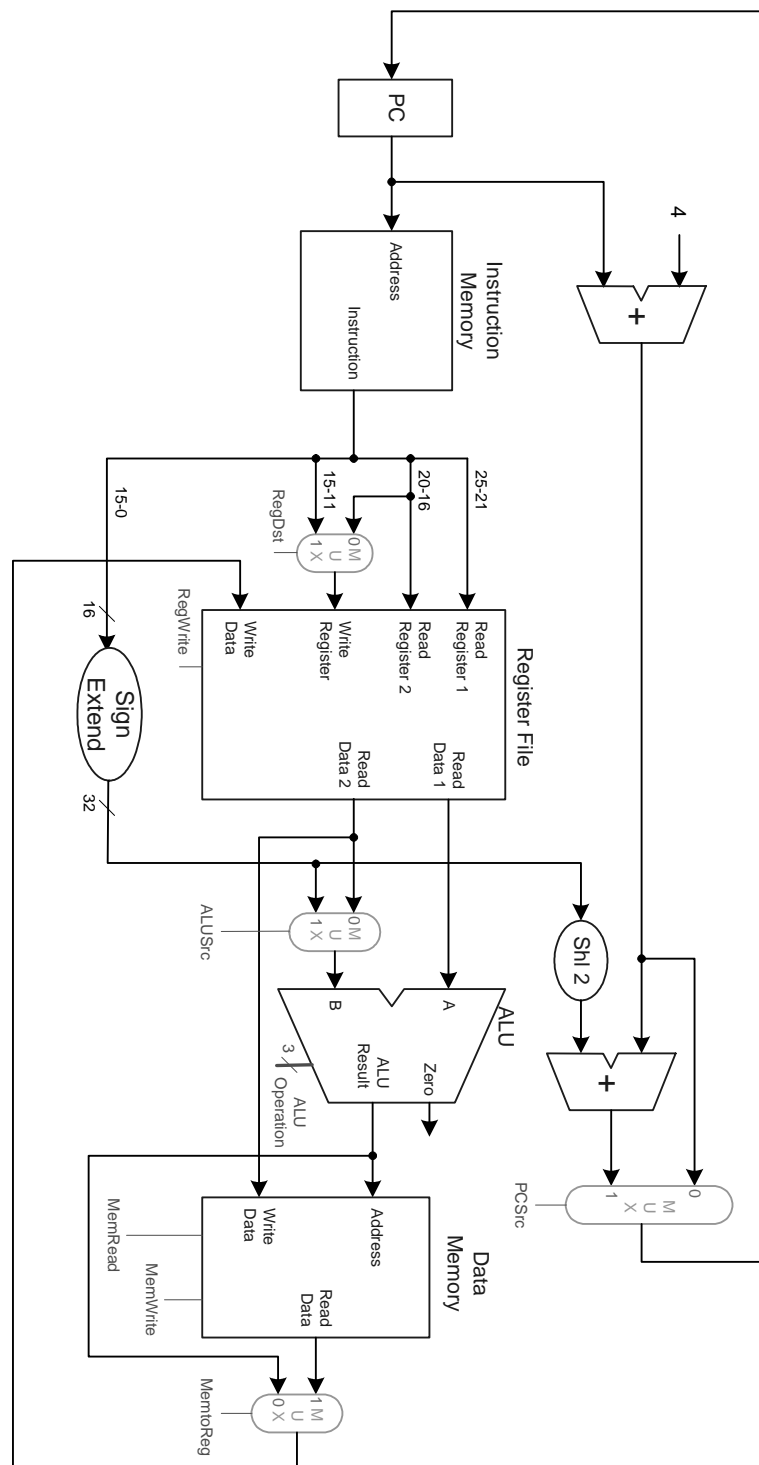
مرحله ی پنجم: ادغام دو بخش سوم و چهارم



مرحله ی ششم: طراحی مسیر داده برای دستور *beq*



مرحله‌ی هفتم: ادغام دو بخش پنجم و ششم



طراحی واحد کنترل

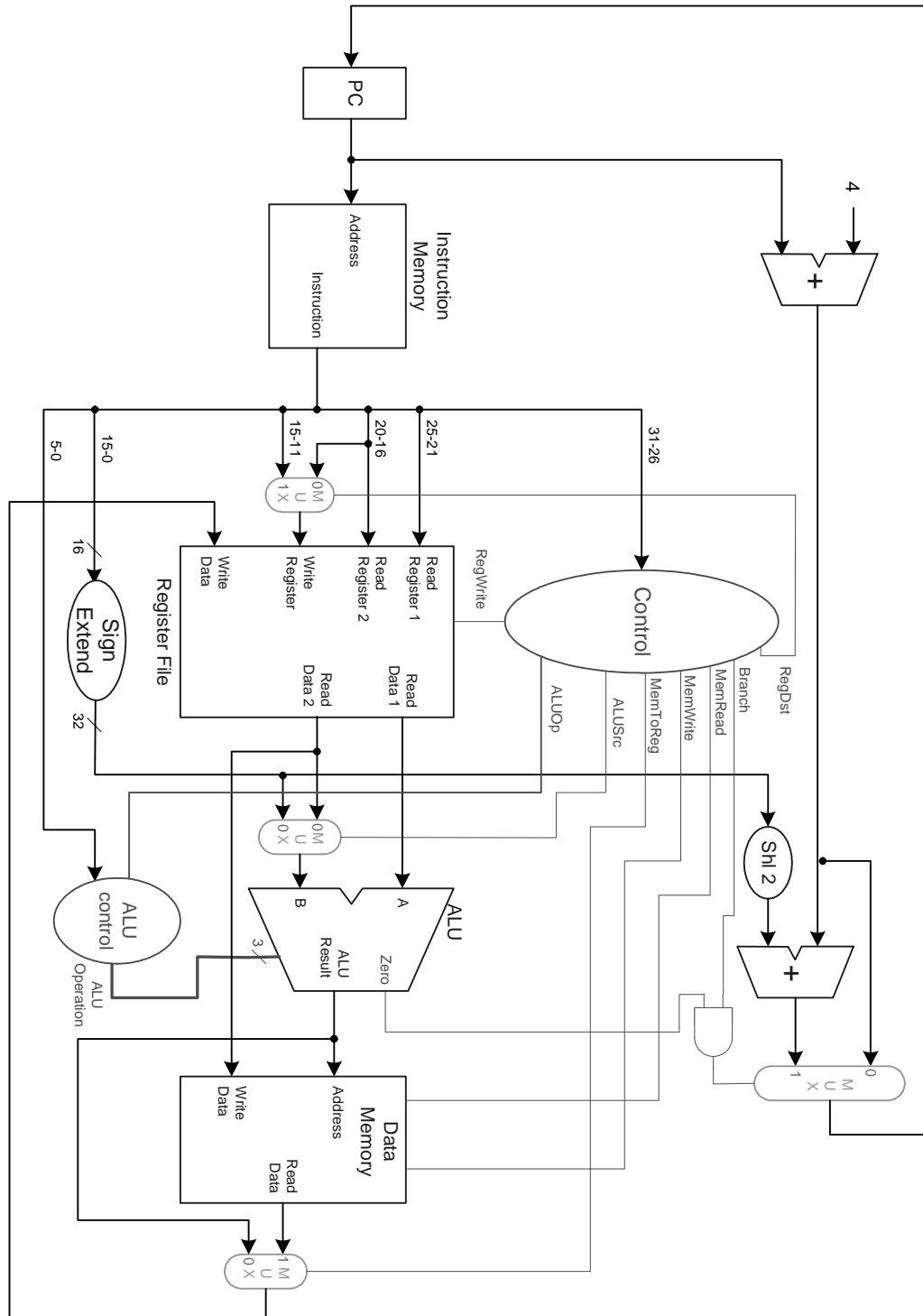
ALU Control

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





▪ 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.

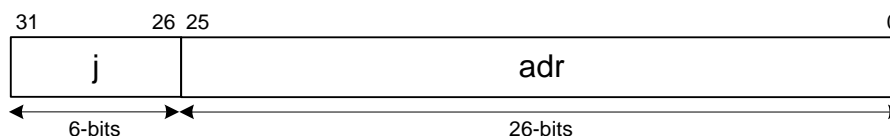
- نحوه‌ی اجرای دستورات *R-Type* (مثال: `add $1, $2, $3`)
  - دستور از حافظه‌ی دستور، واکنشی (Fetch) می‌شود و PC افزایش می‌یابد.
  - دو رجیستر (`$2, $3`) از رجیستر فایل خوانده می‌شود.
  - ALU روی داده‌های خوانده شده از رجیسترفایل عمل می‌کند (براساس بیت‌های ۰ تا ۵ دستور) و خروجی ALU تولید می‌شود.
  - نتیجه‌ی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

- نحوه‌ی اجرای دستور *lw* (مثال: `lw $1, offset($2)`)
  - دستور از حافظه‌ی دستور، واکنشی (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، واکنشی (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، واکنشی (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 می‌شود.
  - 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:

- ۴ بیت رتبه‌ی بالای  $PC+4$  (بیت‌های 31-28)
- ۲۶ بیت آدرس موجود در دستور (بیت‌های 25-0)
- ۲ بیت 00