به نام خدا

درس معماری کامپیوتر
نیم‌سال دوم ۰۴-۰۳
استاد: دکتر اسدی

دانشکده مهندسی کامپیوتر

پاسخنامه تمرین سری ششم

---

۱. (آ) CPI پردازنده P1 به صورت زیر محاسبه می‌شود:

$$CPI = 0.4 \times 6 + 0.2 \times 6 + 0.3 \times 2 + 0.1 \times 2 = 4.4$$

(ب) CPI پردازنده P2 به صورت زیر محاسبه می‌شود:

$$CPI = 0.4 \times 10 + 0.2 \times 10 + 0.3 \times 6 + 0.1 \times 6 = 8.4$$

(ج) پردازنده P1، 1.05 برابر سریع‌تر از P2 است.

$$ExecutionTime_{P1} = Instructions \times CPI_{P1} \times ClockCycleTime$$

$$ExecutionTime_{P2} = Instructions \times CPI_{P2} \times \frac{ClockCycleTime}{2}$$

$$\frac{ExecutionTime_{P1}}{Execution_Time_{P2}} = \frac{instructions \times CPI_{P1} \times ClockCycleTime}{instructions \times CPI_{P2} \times \frac{ClockCycleTime}{2}} = \frac{4.4 \times 1}{8.4 \times \frac{1}{2}} = \frac{4.4}{4.2} = 1.05$$

(د) بهبود ALU گزینه بهتری است.
با بهبود در قسمت ALU، CPI پردازنده P1 برای برنامه A به صورت زیر محاسبه می‌شود:

$$CPI_{ALU} = 0.4 \times 6 + 0.2 \times 6 + 0.3 \times \frac{2}{2} + 0.1 \times \frac{2}{2} = 4$$

با بهبود LSU، CPI پردازنده P1 برای برنامه A به صورت زیر محاسبه می‌شود:

$$CPI_{LSU} = 0.4 \times \frac{6}{2} + 0.2 \times (6 \times 2) + 0.3 \times 2 + 0.1 \times 2 = 4.4$$

همان‌طور که مشاهده کردیم، CPI_{ALU} مقدار کمتری نسبت به CPI_{LSU} داشت و در نتیجه بهبود در آن، نتیجه بهتری دارد.

۲. (آ) برای پیاده‌سازی این دستور، تنظیم میکنیم که ALU عملیات تفریق را انجام دهد و با استفاده از MSB، اینکه $rs بزرگتر مساوی است و یا $rt را مشخص میکنیم. حال از یک مالتی پلکسر استفاده میکنیم که ورودی ۰ آن $rs، ورودی ۱ آن $rt و سیگنال کنترلی آن نیز بیت MSB نتیجه است. در نهایت نیز از یک مالتی پلکسر دیگر استفاده میکنیم که ورودی ۰ آن، خروجی مالتی پلکسر قبلی که به write data در رجیسترفایل می‌رفت باشد و ورودی ۱ آن نیز خروجی مالتی پلکسری که قبل‌تر اشاره کردیم. برای بیت کنترلی نیز از یک سیگنال is_max که از واحد کنترل می‌آید و در زمان اجرای این دستور یک می‌شود، استفاده میکنیم.

(ب) کلاک اول:
در این کلاک مقدار PC برابر با PC+4می‌شود و همچنین مقدار IR برابر با Mem[PC] می‌شود.
سیگنال‌ها:
IorD=0, MemRead=1, MemWrite=0, IRWrite=1, RegWrite=0, ALUsrcA=0, ALUsrcB=01, PCWrite=1, ALUop=00, PCsource=00, MemtoReg=X, RegDst=X, is_max=X
کلاک دوم:
در این کلاک مقادیر رجیسترهای مدنظر خوانده و ذخیره می‌شوند، همچنین مقادیر سیگنال کنترلی به طور خاص این دستور مشخص می‌شوند. ALU نیز مشغول محاسبه مقصد پرش احتمالی برای دستور Branch است. سیگنال‌ها:
IorD=X, MemRead=0, MemWrite=0, IRWrite=0, RegWrite=0, ALUsrcA=0, ALUsrcB=11, PCWrite=0, ALUop=00, PCsource=X, MemtoReg=X, RegDst=X, is_max=X
کلاک سوم:
در این کلاک عمل تفریق مورد نظر در ALU انجام می‌شود.
سیگنال‌ها:
IorD=X, MemRead=0, MemWrite=0, IRWrite=0, RegWrite=0, ALUsrcA=1, ALUsrcB=00, PCWrite=0, ALUop=10, PCsource=X, MemtoReg=X, RegDst=X, is_max=X
کلاک چهارم:
در این کلاک WB انجام شده و نتیجه در رجیسترفایل ذخیره می‌شود. سیگنال‌ها:
IorD=X, MemRead=0, MemWrite=0, IRWrite=0, RegWrite=1, ALUsrcA=X, ALUsrcB=X, PCWrite=0, ALUop=X, PCsource=X, MemtoReg=X, RegDst=1, is_max=1

(ج) همانطور که در قبل دیدیم، CPI این دستور برابر با ۴ است.
دستور la عملا معادل ori و lui است که هر کدام ۴ کلاک هستند.
دستور lw شامل ۵ کلاک است.
دستورات sw, add, sub همگی شامل ۴ کلاک هستند.
دستور beq نیز شامل ۳ کلاک است.
از طرفی با توجه به مقادیر ذخیره‌شده، متوجه میشویم که دستور برنچ not taken است و بنابراین داریم:

$$CPI = \frac{6 * 4 + 3 + 2 * 5}{9} = 4.11$$

۳. (آ) در پردازنده single-cycle برای اجرای هر یک از دستورات به یک چرخه ساعت نیاز است و طول هر چرخه ساعت برابر با زمان اجرای طولانی‌ترین دستور یعنی lw است. بنابراین زمان چرخه ساعت در پردازنده single-cycle برابر می‌شود با:

$$200 + 50 + 100 + 200 + 50 = 600$$

اما تعداد چرخه‌های ساعت برای اجرای هر دستور در پدازنده multi-cycle بسته به نوع دستور متفاوت است:

J	beq	sw	lw	R	Instruction Type
۳	۳	۴	۵	۴	clock cycle count

همچنین در این پردازنده، طول چرخه ساعت باید به اندازه‌ای باشد که طولانی‌ترین بخش قابل اجرا باشد. بنابراین طول چرخه ساعت برابر با 200nm خواهد بود.

(ب)

$$CPI_{single-cycle} = 1$$

$$CPI_{multi-cycle} = (0.25 \times 5) + (0.1 \times 4) + (0.52 \times 4) + (0.11 \times 3) + (0.02 \times 3) = 4.12$$

(ج) میزان speedup به صورت زیر محاسبه می‌شود:

$$\frac{CPI_{single-cycle} \times clockcycletime_{single-cycle}}{CPI_{multi-cycle} \times clockcycletime_{multi-cycle}} = \frac{1 \times 600}{4,12 \times 200}$$

$$\frac{1 \times 600}{4.12 \times 200} \approx 0.73$$

بنابراین تسریعی حاصل نمی‌شود.

۴. اگر فرض کنیم $x\%$ از دستورات از نوع X، $y\%$ از دستورات از نوع Y و $z\%$ از دستورات از نوع Z هستند، مقدار CPI برای هر یک از ۳ پردازنده برابر است با:

$$CPI_{p1} = 5x + 10y + 3z$$

$$CPI_{p2} = 3x + 7y + 2z$$

$$CPI_{p3} = 2x + 5y + 1z$$

از آنجایی که برنامه محک روی پردازنده $P2$ به میزان 1.8 برابر کوتامتر نسبت به پردازنده $P1$ اجرا می‌شود، خواهیم داشت:

$$\frac{ExecTime_{p1}}{ExecTime_{p2}} = \frac{\frac{CPI_{p1}}{Frequency_{p1}}}{\frac{CPI_{p2}}{Frequency_{p2}}} = \frac{\frac{5x+10y+3z}{1}}{\frac{3x+7y+2z}{1.2}} = 1.8$$

$$\Rightarrow 10x + 20y + 6z = 9x + 21y + 6z \quad \Rightarrow \quad x = y$$

همچنین چون اجرای برنامه محک روی پردازنده $P3$ به میزان 1.875 برابر کوتامتر نسبت به پردازنده $P2$ اجرا می‌شود، خواهیم داشت:

$$\frac{ExecTime_{p2}}{ExecTime_{p3}} = \frac{\frac{CPI_{p2}}{Frequency_{p2}}}{\frac{CPI_{p3}}{Frequency_{p3}}} = \frac{\frac{3x+7y+2z}{1.2}}{\frac{2x+5y+1z}{1.5}} = 1.875$$

$$\Rightarrow 15x + 35y + 10z = 15x + 37.5y + 7.5z \quad \Rightarrow \quad y = z$$

همچنین از آنجایی که برنامه محک تنها از دستورات X, Y, Z تشکیل شده است، داریم:

$$x + y + z = 1$$

بنابراین کافیست سه معادله سه مجهول بالا را حل کنیم که در نتیجه خواهیم داشت:

$$x = 0.33, \quad y = 0.33, \quad z = 0.33$$

بنابراین سه نوع دستور X, Y, Z به‌صورت برابر در بنچ‌مارک به کار رفته‌اند.

تنها بخش قابل تسریع در سری پردازنده مذکور، مسیر بحرانی آن است. بیشینه تسریع فرکانسی که این سری پردازنده داشته است برابر 1.5 بوده که برای این اتفاق، باید T_{total} به اندازه $\frac{1}{3}$ کاهش یابد. با توجه به داده سوال، بهترین کران بالایی که می‌توان برای T_{limit} بدست آورد برابر $\frac{2}{3}$ T حالت ابتدایی یعنی $\frac{2}{3} \times \frac{1}{F}$ خواهد بود.

۵. (آ) از آنجا که FSM دارای ۱۲ فلیپ‌فلاپ است، یعنی ۱۲ بیت برای نمایش وضعیت قبلی و فعلی نیاز است. اگر از یک ROM استفاده کنیم خواهیم داشت:

$$\#\ of\ inputs = 8(opcode) + 12(stateregister) = 20bits$$
$$\#\ of\ outputs = 30(controlsignals) + 12(nextstate) = 42bits$$
$$size\ of\ ROM = 2^{20} \times 42bits = 42Mb$$

(ب) از ROM1 برای بدست آوردن حالت بعدی و از ROM2 برای بدست آوردن سیگنال‌های خروجی استفاده می‌کنیم.

$$\#\ of\ inputs(ROM1) = 8(opcode) + 12(stateregister) = 20bits$$
$$\#\ of\ outputs(ROM1) = 12(nextstate) = 12bits$$
$$size\ of\ ROM1 = 2^{20} \times 12bits = 12Mb$$
$$\#\ of\ inputs(ROM2) = 12(stateregister) = 12bits$$
$$\#\ of\ outputs(ROM2) = 30(controlsignals) + 12(nextstate) = 42bits$$
$$size\ of\ ROM = 2^{12} \times 42bits = 168Kb$$

(ج) اگر هر دستور به ۸ میکرودستور تقسیم شود، نیاز است که ۳ بیت دیگر به تعداد بیت‌های وضعیت (فعلی و بعدی) اضافه شود.

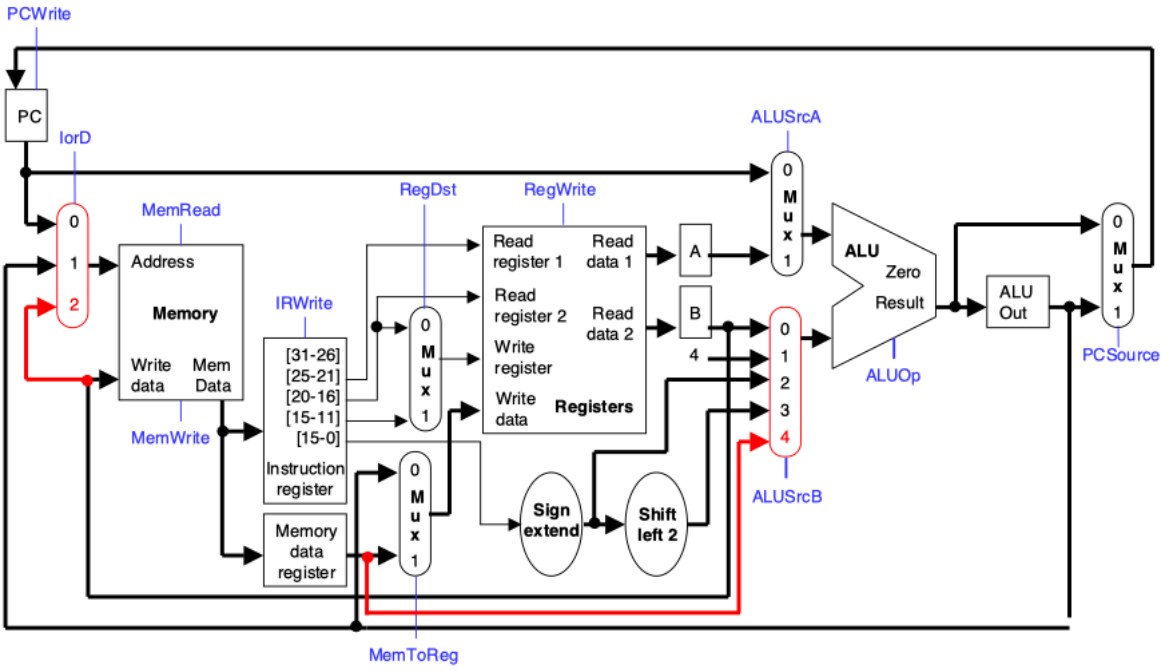
حل مجدد بخش الف:

$$\#\ of\ inputs = 8(opcode) + 12(stateregister) + 3(u - instructionstate) = 23bits$$
$$\#\ of\ outputs = 30(controlsignals) + 12(nextstate) + 3(u - instructionstate) = 45bits$$
$$size\ of\ ROM = 2^{23} \times 45bits = 360Mb$$

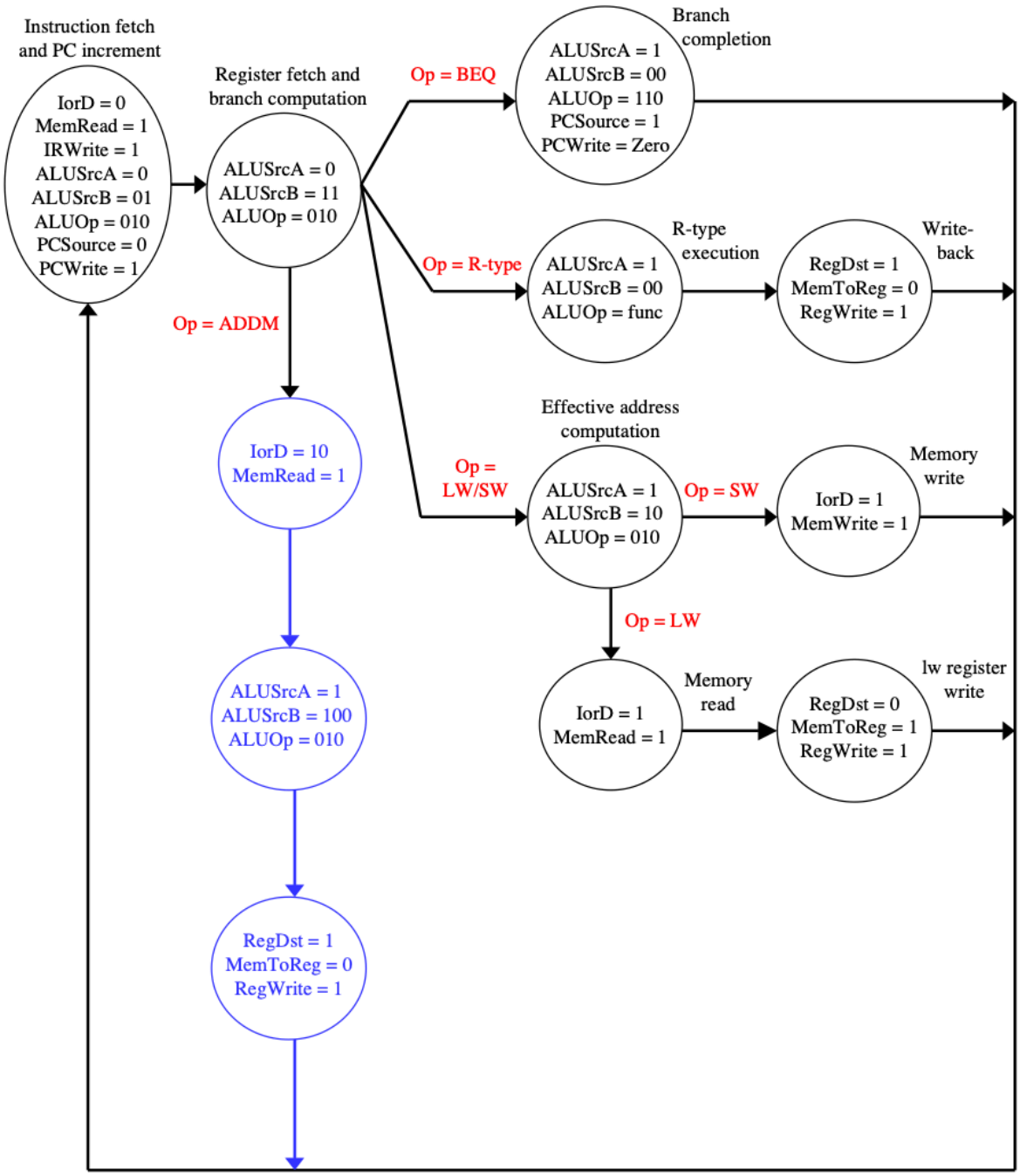
حل مجدد بخش ب:

$$\#\ of\ inputs(ROM1) = 8(opcode) + 12(stateregister) + 3(u - instructionstate) = 23bits$$
$$\#\ of\ outputs(ROM1) = 12(nextstate) + 3(u - instructionstate) = 15bits$$
$$size\ of\ ROM1 = 2^{23} \times 15bits = 120Mb$$
$$\#\ of\ inputs(ROM2) = 12(stateregister) + 3(u - instructionstate) = 15bits$$
$$\#\ of\ outputs(ROM2) = 30(controlsignals) + 12(nextstate) + 3(u - instructionstate) = 45bits$$
$$size\ of\ ROM = 2^{15} \times 45bits = 1440Kb$$

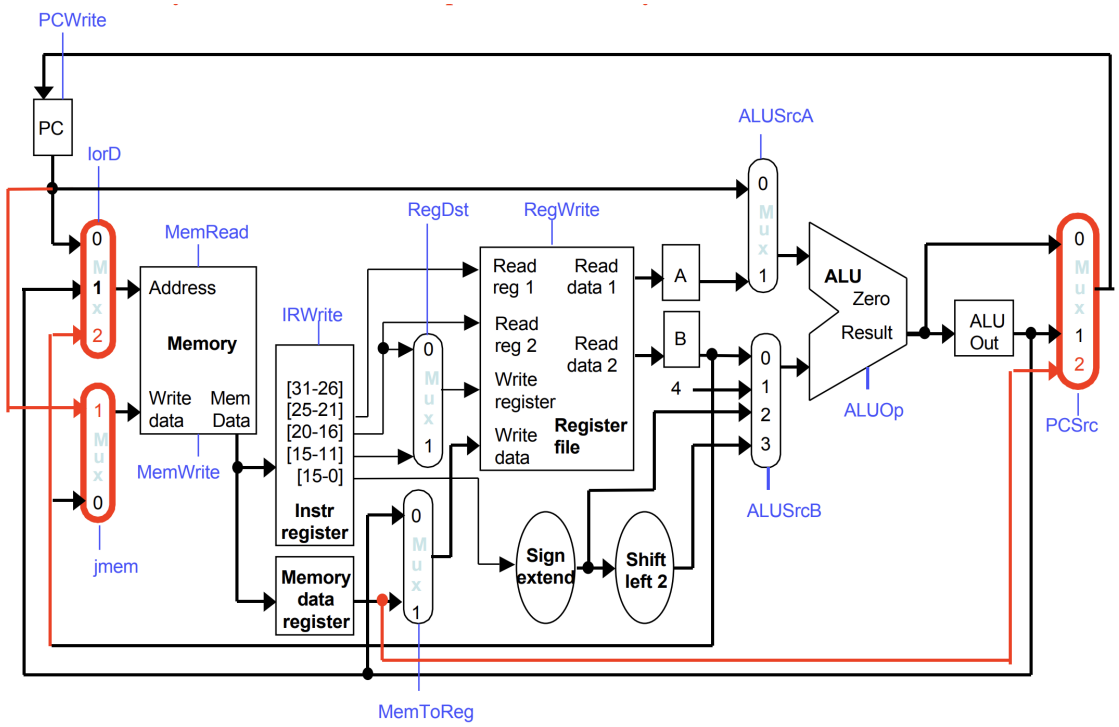
۶. الف) ما رجیستر میانی B را به واحد حافظه متصل کرده‌ایم تا بتوانیم از آدرس rt داده بخوانیم. همچنین، مقدار MDR (که شامل مقدار Mem[rt] خواهد بود) را به واحد ALU ورودی داده‌ایم تا با رجیستر rs جمع شود. این تغییرات، احتمالاً ساده‌ترین مجموعه تغییرات ممکن هستند.



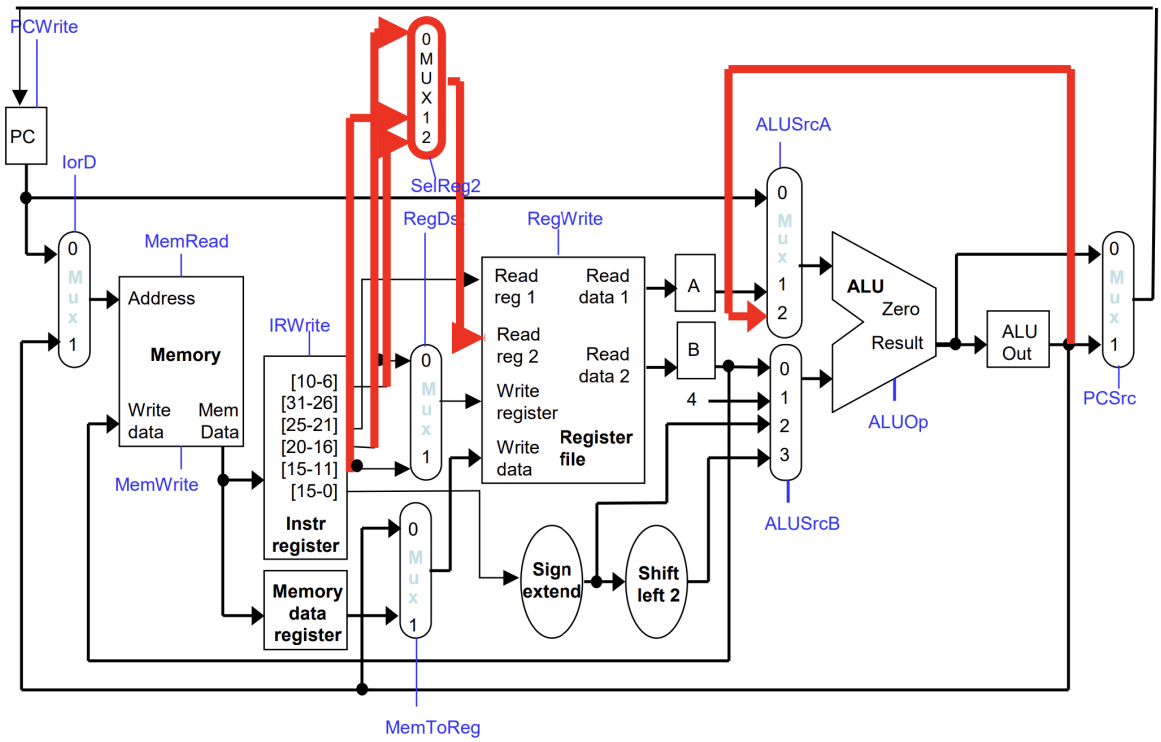
ب) با توجه به مسیر داده‌ای جدید که در صفحهٔ قبل نشان داده شده است، پیاده‌سازی دستور addm کار سختی نیست. دو سیکل اول مانند سایر دستورات پیش می‌روند: ابتدا دستور خوانده می‌شود و سپس مقادیر رجیسترها خوانده می‌شوند. در سیکل سوم، می‌توانیم از مقدار B (که دادهٔ رجیستر rs را در خود دارد) به عنوان آدرس حافظه برای خواندن داده استفاده کنیم. سپس مقدار Mem[rt] در سیکل بعدی با رجیستر rs جمع می‌شود. توجه کنید که مقدار A همچنان دادهٔ مربوط به rs را نگه داشته است، چرا که ورودی‌های فایل رجیستر تغییر نکرده‌اند. در نهایت، نتیجه مانند یک دستور نوع R به رجیستر مقصد بازنویسی می‌شود.


Instruction fetch and PC increment
IorD = 0
MemRead = 1
IRWrite = 1
ALUSrcA = 0
ALUSrcB = 01
ALUOp = 010
PCSource = 0
PCWrite = 1
Register fetch and branch computation
ALUSrcA = 0
ALUSrcB = 11
ALUOp = 010
Op = BEQ
Branch completion
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 110
PCSource = 1
PCWrite = Zero
Op = R-type
R-type execution
ALUSrcA = 1
ALUSrcB = 00
ALUOp = func
Write-back
RegDst = 1
MemToReg = 0
RegWrite = 1
Op = ADDM
IorD = 10
MemRead = 1
ALUSrcA = 1
ALUSrcB = 100
ALUOp = 010
RegDst = 1
MemToReg = 0
RegWrite = 1
Effective address computation
Op = LW/SW
ALUSrcA = 1
ALUSrcB = 10
ALUOp = 010
Op = SW
Memory write
IorD = 1
MemWrite = 1
Op = LW
Memory read
IorD = 1
MemRead = 1
lw register write
RegDst = 0
MemToReg = 1
RegWrite = 1

۷. (آ) تغییرات اعمال شده برای دستور jmem بصورت زیر است:

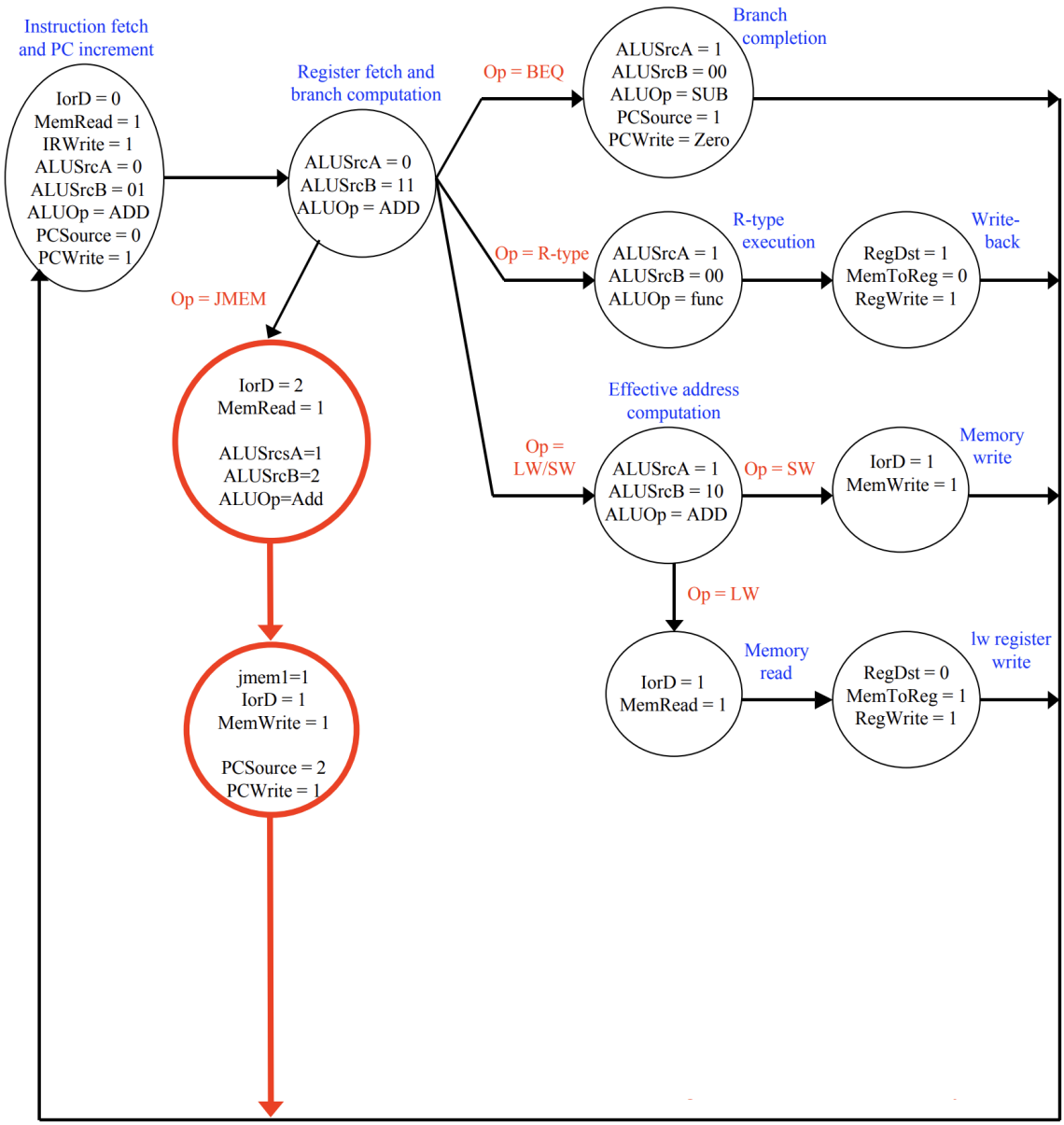


(ب) تغییرات اعمال شده برای دستور max4 به صورت زیر است:

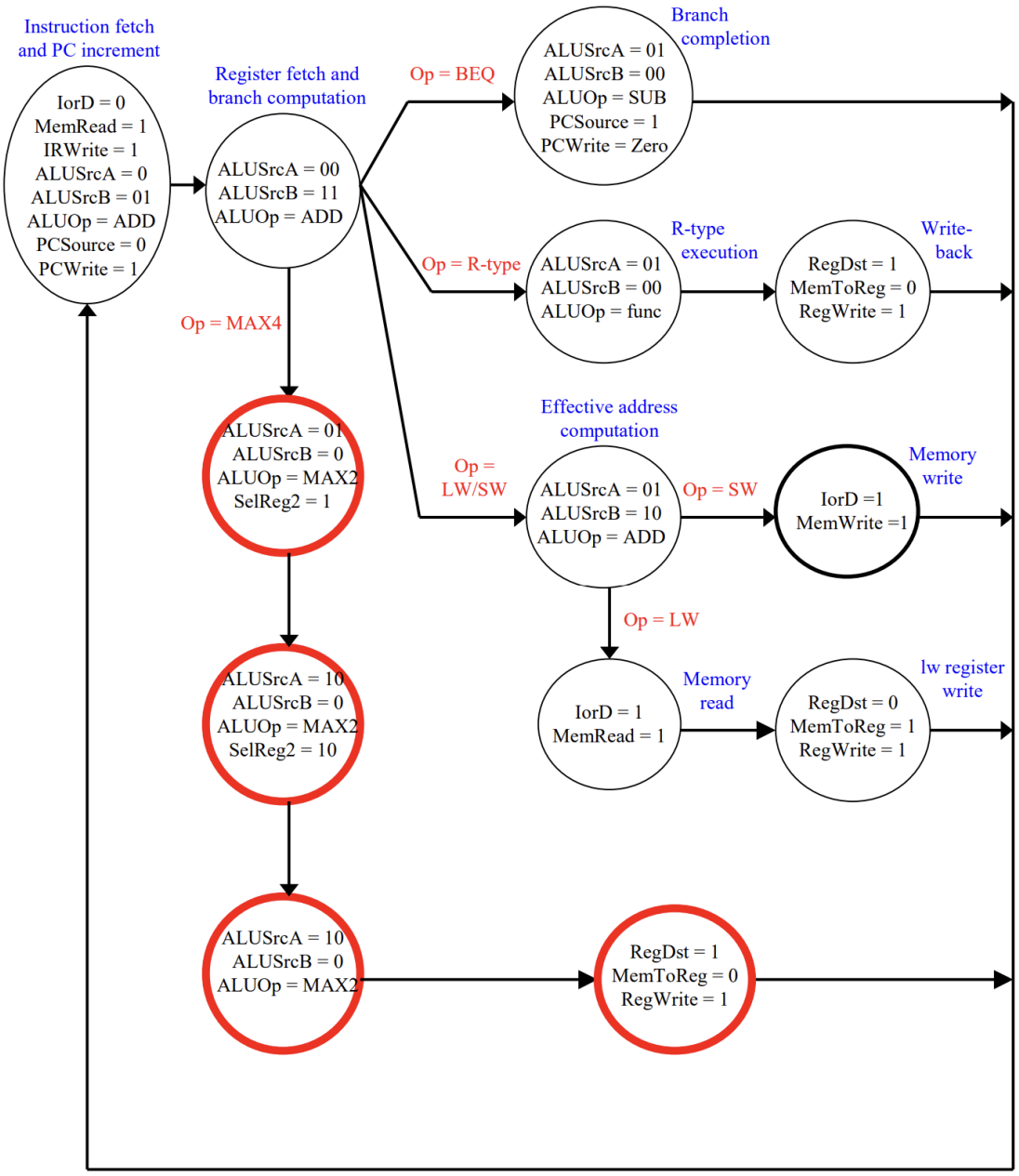


(ج)
- دستور اول: راه‌های پیاده‌سازی دیگری نیز برای این دستور وجود دارد. راهی که ما استفاده کردیم، آن را در ۴ سیکل انجام می‌دهد. همه‌ی راه‌حل‌ها باید مسیر داده‌ای از ثبات PC به پورت Write data در حافظه و از پورت MemData در حافظه به PC اضافه شود. از آنجا که آدرس دستور Store دارای یک مقدار فوری است، باید از ALU (در سیکل ۳) استفاده کنیم قبل از اینکه عملیات Store را در سیکل ۴ انجام دهیم.

مقدار PC+4 در سیکل ۱ در ثبات PC ذخیره شده، بنابراین می‌توان آن را در هر زمانی پس از تولید آدرس، ذخیره کرد. برای اجرای این عملیات در ۴ سیکل، باید عمل Load در سیکل ۳ انجام شود. مسیر داده‌ای از ثبات B به پورت آدرس حافظه اضافه شود. انجام Load پیش از Store ممکن است چون تضمین شده که این دو آدرس با هم هم‌پوشانی ندارند. برای جلوگیری از بازنویسی مقدار PC+4 ، مقدار بارگذاری‌شده در ثبات MDR نگه داشته می‌شود و در سیکل ۴ به عقب نوشته می‌شود. ماشین حالت متناهی آن به صورت زیر است:



- دستور دوم: کافی است دستور را در ۶ سیکل انجام دهیم، در ۲ سیکل اول که همانند بقیه دستورات است. در سیکل های ۳، ۴ و ۵ باید مقایسه بین ثبات ها را داشته باشیم، چون طبق سوال ALU مقایسه ۲ به ۲ را انجام می دهد و برای این ۴ ثبات نیاز به ۳ تا مقایسه داریم و در سیکل ۶ که آخرین سیکل است، مقدار محاسبه شده را در ثبات مقصد ذخیره می کنیم. ماشین حالت متناهی آن به صورت زیر است:


Instruction fetch and PC increment
IorD = 0
MemRead = 1
IRWrite = 1
ALUSrcA = 0
ALUSrcB = 01
ALUOp = ADD
PCSource = 0
PCWrite = 1
Register fetch and branch computation
ALUSrcA = 00
ALUSrcB = 11
ALUOp = ADD
Op = BEQ
Branch completion
ALUSrcA = 01
ALUSrcB = 00
ALUOp = SUB
PCSource = 1
PCWrite = Zero
Op = R-type
R-type execution
ALUSrcA = 01
ALUSrcB = 00
ALUOp = func
RegDst = 1
MemToReg = 0
RegWrite = 1
Write-back
Op = MAX4
ALUSrcA = 01
ALUSrcB = 0
ALUOp = MAX2
SelReg2 = 1
Effective address computation
Op = LW/SW
ALUSrcA = 01
ALUSrcB = 10
ALUOp = ADD
Op = SW
IorD =1
MemWrite =1
Memory write
Op = LW
ALUSrcA = 10
ALUSrcB = 0
ALUOp = MAX2
SelReg2 = 10
IorD = 1
MemRead = 1
Memory read
RegDst = 0
MemToReg = 1
RegWrite = 1
lw register write
ALUSrcA = 10
ALUSrcB = 0
ALUOp = MAX2
RegDst = 1
MemToReg = 0
RegWrite = 1