## حل تمرین اختیاری دو

بلوک دیاگرام مسیر داده و کنترل پردازنده MIPS را در شکل ۱ مشاهده می‌کنید. در طراحی این پردازنده فرض این بوده است که قرار است تنها دستورالعمل‌های add، sub، and، or، slt، lw، sw، beq و j اجرا شود. عملیات کنترل بلوک دیاگرام شکل ۱ توسط تعدادی سیگنال کنترلی انجام می‌شود که در جدول‌های ۱ و ۲ آمده است.

۱- توجه دارید که جدول ۲ سیگنال(های) کنترلی موردنیاز برای پیاده‌سازی دستورالعمل j را ندارد. این جدول را طوری کامل کنید که این سیگنال(های) کنترلی را نیز تولید کند.

۲- اگر بخواهیم دستوراالعمل‌های addi، andi، ori و slti را هم به مجموعه دستورالعمل‌ها اضافه کنیم، چه تغییراتی باید در شکل و جداول بدهیم؟

۳- اگر بخواهیم دستورالعمل bne را هم به مجموعه دستورالعمل‌ها اضافه کنیم، چه تغییراتی باید در شکل و جداول بدهیم؟

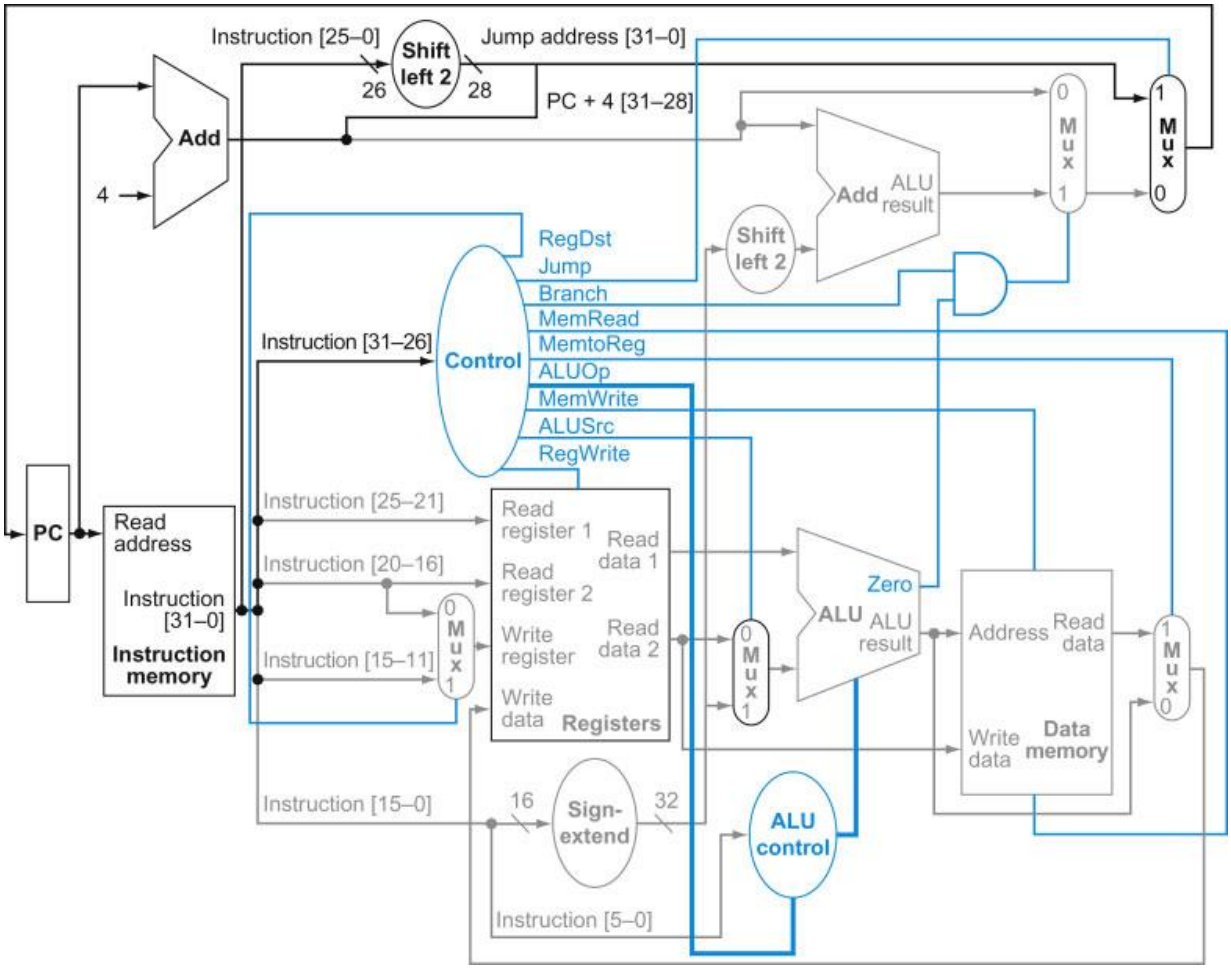
۴- اگر بخواهیم دستورالعمل jr را هم به مجموعه دستورالعمل‌ها اضافه کنیم، چه تغییراتی باید در شکل و جداول بدهیم؟

جدول ۱- شرح ارتباط سیگنال‌های واحد ALU Control در شکل ۱

| 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 |

جدول ۲- شرح ارتباط سیگنال‌های واحد Control در شکل ۱

| Input or output | Signal name | R-format | lw | sw | beq |
|---|---|---|---|---|---|
| Inputs | Op5 | 0 | 1 | 1 | 0 |
| | Op4 | 0 | 0 | 0 | 0 |
| | Op3 | 0 | 0 | 1 | 0 |
| | Op2 | 0 | 0 | 0 | 1 |
| | Op1 | 0 | 1 | 1 | 0 |
| | Op0 | 0 | 1 | 1 | 0 |
| Outputs | RegDst | 1 | 0 | X | X |
| | ALUSrc | 0 | 1 | 1 | 0 |
| | MemtoReg | 0 | 1 | X | X |
| | RegWrite | 1 | 1 | 0 | 0 |
| | MemRead | 0 | 1 | 0 | 0 |
| | MemWrite | 0 | 0 | 1 | 0 |
| | Branch | 0 | 0 | 0 | 1 |
| | ALUOp1 | 1 | 0 | 0 | 0 |
| | ALUOp0 | 0 | 0 | 0 | 1 |




شکل ۱- بلوک دیاگرام مسیر داده و کنترل پردازنده ساده MIPS

حل سوالات:

۱- سیگنال کنترلی که در جدول ۲ نیامده است، سیگنال jump است. بنابراین سطر و ستون زیر باید به آن جدول اضافه شود:

| Input or Output | Signal Name | R-format | lw | sw | beq | j |
|---|---|---|---|---|---|---|
| Inputs | Op5 | 0 | 1 | 1 | 0 | 0 |
| | Op4 | 0 | 0 | 0 | 0 | 0 |
| | Op3 | 0 | 0 | 1 | 0 | 0 |
| | Op2 | 0 | 0 | 0 | 1 | 0 |
| | Op1 | 0 | 1 | 1 | 0 | 1 |
| | Op0 | 0 | 1 | 1 | 0 | 0 |
| Outputs | RegDst | 1 | 0 | X | X | X |
| | ALUSrc | 0 | 1 | 1 | 0 | X |
| | MemtoReg | 0 | 1 | X | X | X |
| | RegWrite | 1 | 1 | 0 | 0 | 0 |
| | MemRead | 0 | 1 | 0 | 0 | 0 |
| | MemWrite | 0 | 0 | 1 | 0 | 0 |
| | Branch | 0 | 0 | 0 | 1 | X |
| | ALUOp1 | 1 | 0 | 0 | 0 | X |
| | ALUOp2 | 0 | 0 | 0 | 1 | X |
| | Jump | 0 | 0 | 0 | 0 | 1 |

۲- برای اجرای این چهار دستور باید واحد ALU Control را تغییر دهیم. قبلا وقتی ALUOp=01، بسته به مقدار فیلد funct (بیت‌های صفر تا ۵ دستورالعمل) عملیات داخل ALU تغییر می‌کرد. اما الان بیت‌های صفر تا ۵ معنای func نمی‌دهند، بنابراین باید راه دیگری پیدا کنیم. یک راه این است که حالت ALUOp=11 را به جدول ۱ اضافه کنیم و در حالتی که ALUOp=11، به جای شش بیت funct، از شش بیت op (بیت‌های ۲۶ تا ۳۲ دستورالعمل) برای تعیین عملکرد ALU استفاده کنیم. بنابراین لازم است این شش بیت را هم به ورودی ALU Control بدهیم.

| opcode | ALUOp | Operation | funct | ALU function | ALU control |
|---|---|---|---|---|---|
| lw | 00 | load word | XXXXXX | add | 0010 |
| sw | 00 | store word | XXXXXX | add | 0010 |
| beq | 01 | branch equal | XXXXXX | subtract | 0110 |
| R-type | 10 | add | 100000 | add | 0010 |
| | | subtract | 100010 | subtract | 0110 |
| | | AND | 100100 | AND | 0000 |
| | | OR | 100101 | OR | 0001 |
| | | set-on-less-than | 101010 | set-on-less-than | 0111 |
| **opcode** | **ALUOp** | **Operation** | **op** | **ALU function** | **ALU control** |
| addi | 11 | add | 001000 | add | 0010 |
| slti | 11 | subtract | 001010 | subtract | 0110 |
| andi | 11 | and | 001100 | and | 0000 |
| ori | 11 | or | 111011 | or | 0001 |

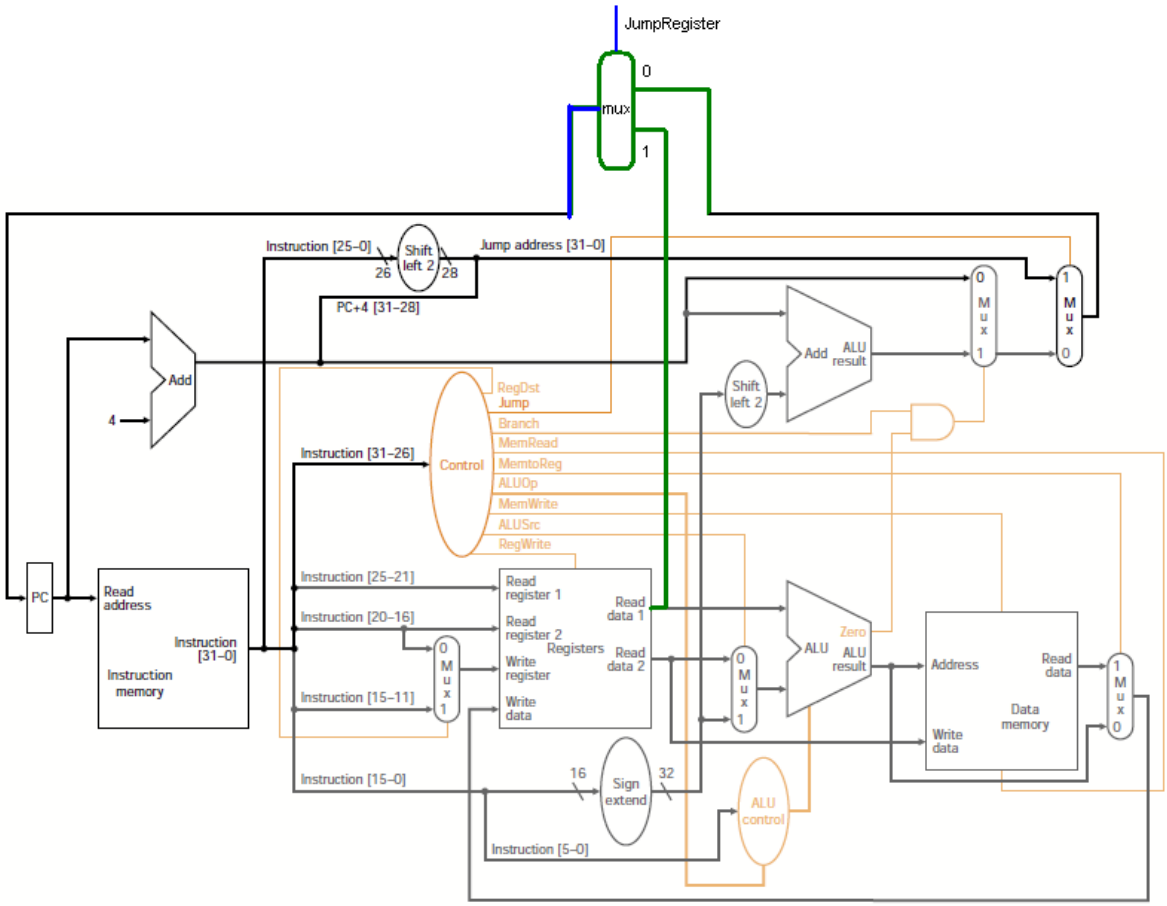
۳- اگر بخواهیم bne را هم اضافه کنیم، باید این امکان را به شکل اضافه کنیم که در صورتی که دستور bne بود و خروجی Zero واحد محاسباتی اصلی (ALU پایین شکل) صفر بود (یعنی مقدار دو ثبات نامساوی بودند)، مقدار PC با شیفت‌یافته ۱۶ بیت کم‌ارزش ثبات دستورالعمل جمع شود. بنابراین لازم است یک سیگنال کنترلی دیگر هم در واحد Control تولید شود که نام آن را BranchNE می‌گذاریم و آن را با Zero' به یک گیت AND می‌دهیم و خروجی این گیت را با خروجی گیت AND قبلی OR می‌کنیم و به ورودی MUX می‌دهیم.
جدول ۲ هم به شکل زیر تغییر می‌کند.

| Input or Output | Signal Name | R-format | lw | sw | beq | j | addi | slti | andi | ori | bne |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Inputs | Op5 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| | Op4 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| | Op3 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| | Op2 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| | Op1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| | Op0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| Outputs | RegDst | 1 | 0 | X | X | X | 0 | 0 | 0 | 0 | X |
| | ALUSrc | 0 | 1 | 1 | 0 | X | 1 | 1 | 1 | 1 | 0 |
| | MemtoReg | 0 | 1 | X | X | X | 0 | 0 | 0 | 0 | X |
| | RegWrite | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 |
| | MemRead | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | MemWrite | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | Branch | 0 | 0 | 0 | 1 | X | 0 | 0 | 0 | 0 | 0 |
| | BranchNE | 0 | 0 | 0 | 0 | X | 0 | 0 | 0 | 0 | 1 |
| | ALUOp1 | 1 | 0 | 0 | 0 | X | 1 | 1 | 1 | 1 | 0 |
| | ALUOp2 | 0 | 0 | 0 | 1 | X | 0 | 1 | 1 | 1 | 1 |
| | Jump | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |

۴- برای اینکه دستور jr را هم بتوانیم اجرا کنیم، باید این امکان را فراهم کنیم که محتوای یک ثبات خوانده شود و مستقیما وارد PC شود. می‌توانیم یک MUX بعد از MUXای که برای دستور j به شکل اضافه شده بود، قرار بدهیم و یک ورودی را به خروجی MUX مربوط به j اختصاص دهیم و ورودی دیگر را از بانک ثبات بگیریم (Read Data1) و ورودی select را هم در مدار ALU Control تولید کنیم. این ورودی را نمی‌توانیم در Control تولید کنیم، چون در دستور jr مثل بقیه دستورات R-format همه بیت‌های Op صفر هستند و دستور توسط بیت‌های function مشخص می‌شود که اصلا وارد واحد کنترل نمی‌شود.
بنابراین این بار جدول ۱ تغییر خواهد کرد، ضمن اینکه این بار باید سیگنال کنترلی مورد نظر از ترکیب خروجی‌های ALU Control تولید شود.
البته حالا که تعداد MUXهای پرش زیاد شده، می‌توانیم همه را با هم ترکیب کنیم اما برای اینکه فهم تغییرات ساده‌تر باشد، اضافه کردن یک MUX جدید هم اشکالی ندارد.

| opcode | ALUOp | Operation | funct | ALU function | ALU control |
|---|---|---|---|---|---|
| lw | 00 | load word | XXXXXX | add | 0010 |
| sw | 00 | store word | XXXXXX | add | 0010 |
| beq | 01 | branch equal | XXXXXX | subtract | 0110 |
| R-type | 10 | add | 100000 | add | 0010 |
| | | subtract | 100010 | subtract | 0110 |
| | | AND | 100100 | AND | 0000 |
| | | OR | 100101 | OR | 0001 |
| | | set-on-less-than | 101010 | set-on-less-than | 0111 |
| | | jr | 001000 | One of the unused combinations of ALU control | |
| **opcode** | **ALUOp** | **Operation** | **op** | **ALU function** | **ALU control** |
| addi | 11 | add | 001000 | add | 0010 |
| slti | 11 | subtract | 001010 | subtract | 0110 |
| andi | 11 | and | 001100 | and | 0000 |
| ori | 11 | or | 111011 | or | 0001 |

| Input or Output | Signal Name | R-format | lw | sw | beq | j | addi | slti | andi | ori | bne | jr |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Inputs | Op5 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| | Op4 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| | Op3 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| | Op2 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| | Op1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| | Op0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 |
| Outputs | RegDst | 1 | 0 | X | X | X | 0 | 0 | 0 | 0 | X | X |
| | ALUSrc | 0 | 1 | 1 | 0 | X | 1 | 1 | 1 | 1 | 0 | X |
| | MemtoReg | 0 | 1 | X | X | X | 0 | 0 | 0 | 0 | X | X |
| | RegWrite | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| | MemRead | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | MemWrite | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | Branch | 0 | 0 | 0 | 1 | X | 0 | 0 | 0 | 0 | 0 | X |
| | BranchNE | 0 | 0 | 0 | 0 | X | 0 | 0 | 0 | 0 | 1 | X |
| | ALUOp1 | 1 | 0 | 0 | 0 | X | 1 | 1 | 1 | 1 | 0 | X |
| | ALUOp2 | 0 | 0 | 0 | 1 | X | 0 | 1 | 1 | 1 | 1 | X |
| | Jump | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | X |
| | Imm | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | X |
| | J2R | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |


JumpRegister
0
mux
1
Instruction [25–0]
Shift left 2
26
28
Jump address [31–0]
PC+4 [31–28]
Add
4
Add
ALU result
Shift left 2
M u x
RegDst
Jump
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Control
Instruction [31–26]
PC
Read address
Instruction [31–0]
Instruction memory
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
Instruction [15–0]
Instruction [5–0]
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
16
Sign extend
32
ALU
Zero
ALU result
ALU control
Address
Read data
Data memory
Write data