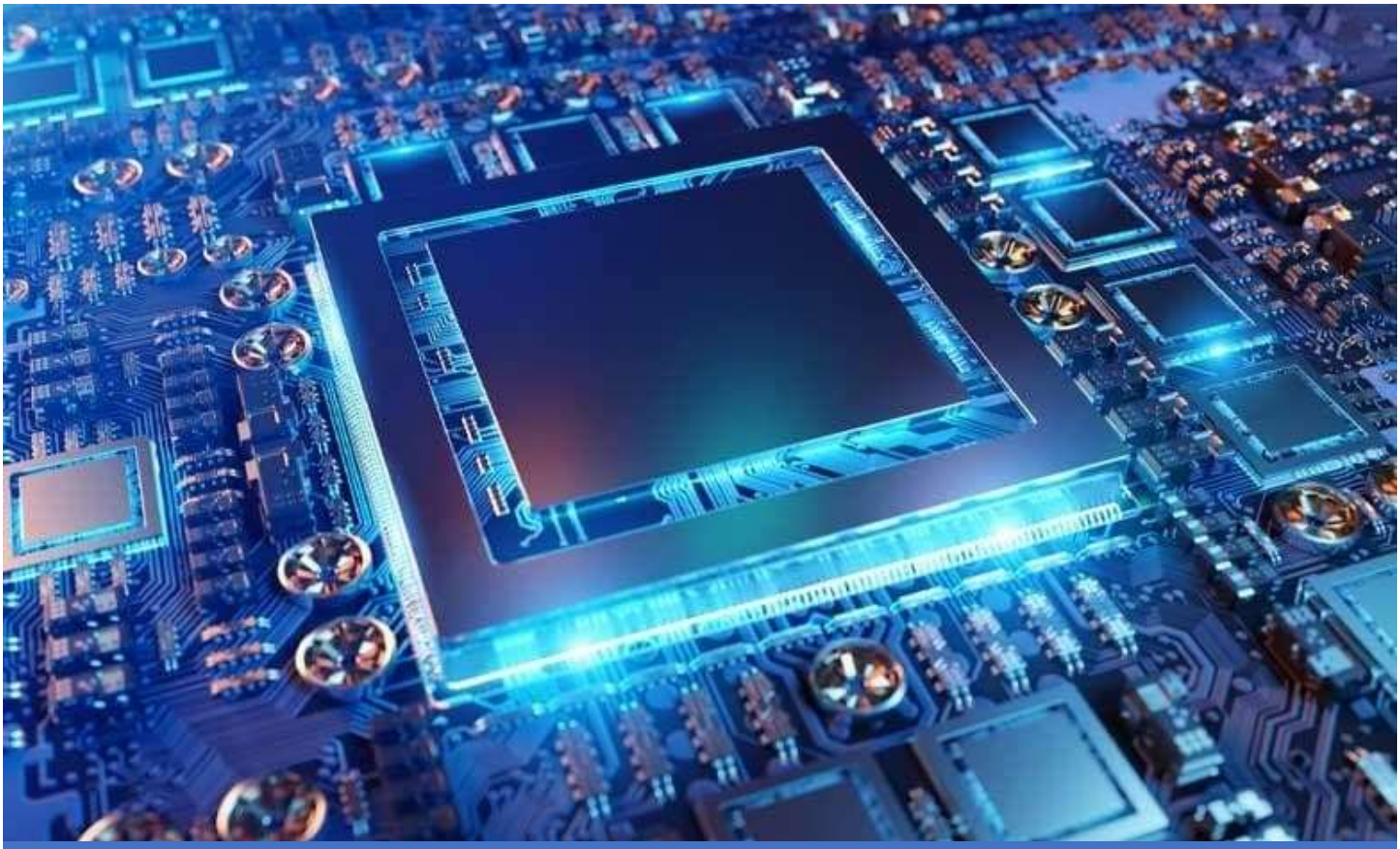
به نام خداوند بخشنده بخشایشگر

عرفان تیموری و محمدرضا منعمیان

شماره دانشجویی : ۴۰۲۱۰۶۶۰۴ و۴۰۲۱۰۵۸۱۳



Practical 4

Erfan Teymouri , Mohammad reza Monemian

4/16/25

computer architecture

شرح پیاده سازی:

در این تمرین با استفاده از ALU ای که در تمرین ۳ طراحی کردیم یک پردازنده کوچک single cycle که از برخی دستورات R-format و I-format پشتیبانی می کند طراحی کردیم.

از ما خواسته شده بود دستورات زیر را پیاده کنیم :

Instruction	Type	Opcode	Funct	Notes
`add`	R-Type	`000000`	`100000`	`rd = rs + rt`
`addi`	I-Type	`001000`	—	`rt = rs + imm` (sign-extended)
`sub`	R-Type	`000000`	`100010`	`rd = rs - rt`
`or`	R-Type	`000000`	`100101`	`rd = rs | rt` (bitwise OR)
`and`	R-Type	`000000`	`100100`	`rd = rs & rt` (bitwise AND)
`xor`	R-Type	`000000`	`100110`	`rd = rs ^ rt` (bitwise XOR)
`sll`	R-Type	`000000`	`000100`	`rd = rs << rt` (logical shift left)
`srl`	R-Type	`000000`	`000110`	`rd = rs >> rt` (logical shift right)
`sra`	R-Type	`000000`	`000111`	`rd = rs >>> rt` (arithmetic shift right, sign-extended)

مشاهده می‌کنیم دستورات ضرب و تقسیم که alu تمرین ۳ قابلیت انجام آن را داشت در این پردازنده پیاده نشده است بنابراین alu ما همه دستورات خواسته شده را می‌تواند در یک کلاک انجام دهد.

ورودی jin داده شده در عمل همان دستور است که به صورت زیر کد گشایی می‌شود (این کار با استفاده از splitter انجام شد ه است) :

اگر R-format باشد :

```
31   26|25  21|20  16|15  11|10  6|5     0
+------+------+------+------+-----+-------+
|opcode|  rs  |  rt  |  rd  |shamt| funct |
+------+------+------+------+-----+-------+
```

اگر I-format باشد:

```
 31  26|25  21|20  16|15              0
+------+------+------+-----------------+
|opcode|  rs  |  rt  |    immediate    |
+------+------+------+-----------------+
```

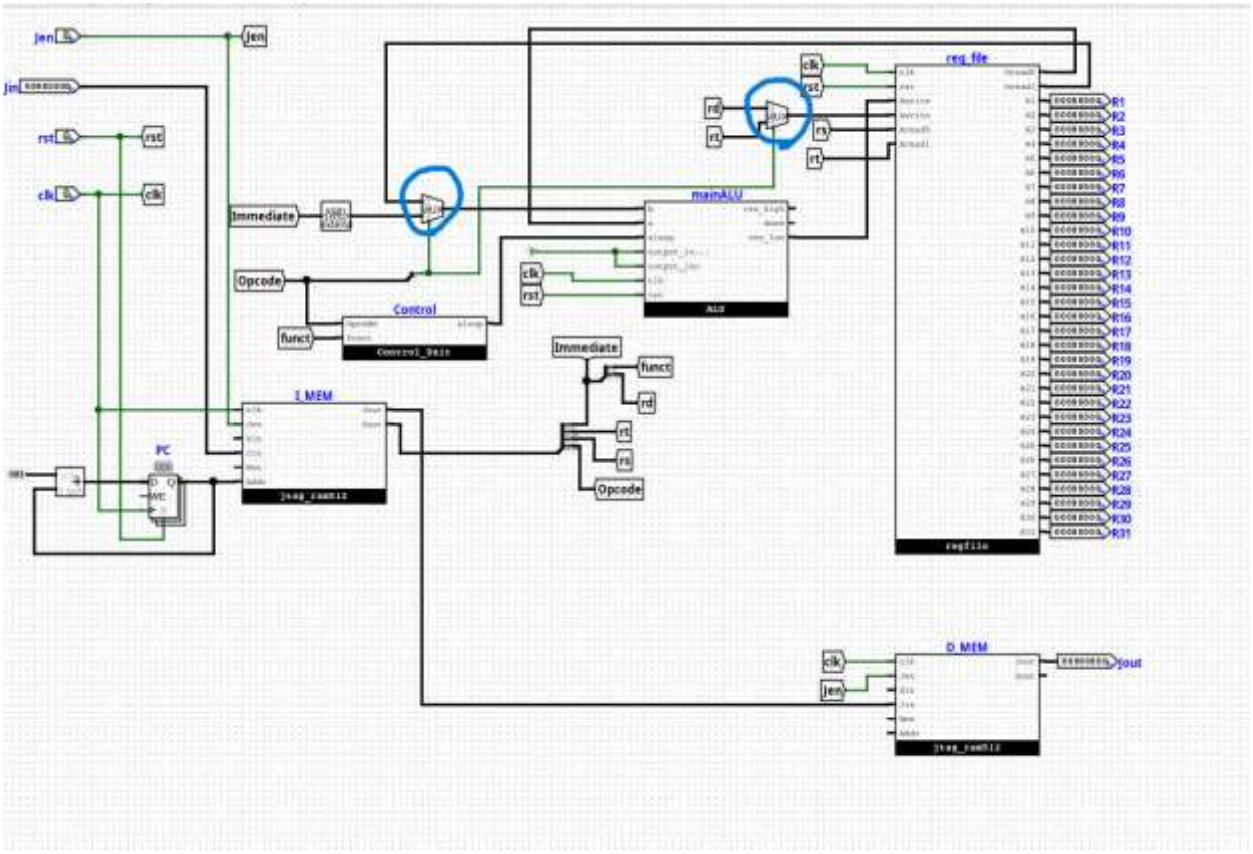
مشاهده می‌کنیم که همیشه ۶ بیت بالایی یعنی بیت ۳۱ تا ۲۶ opcode ما هست که باید با توجه به آن سیگنال ورودی alu را مشخص کنیم.

بیت ۲۵ تا ۲۱ نشان دهنده ثبات rs و بیت ۲۰ تا ۱۶ نشان دهنده ثبات rt است.(توجه داریم در دستورات R-foramt این ثبات یکی از عملگر های عملیات هست و رجیستر مقصد با استفاده از بیت ۱۱ تا ۱۵ مشخص می‌شود یعنی rd و در دستورات I-format رجیستر مقصد هست بنابراین با استفاده از یک MUX اینکه برای رجیستر فایل کدام رجیستر مقصد باشد مشخص شده است)

همچنین مشاهده می‌شود که عملگر دوم عملیات در دستورات R-format و I-format متفاوت است یعنی در اولی رجیستر rt و در دومی مقدار immediate ۱۶ بیتی که در بیت ۰ تا ۱۵ دستور قرار دارد عملگر دوم هست بنابراین این مورد نیز با استفاده از یک MUX مشخص شده است .

بیت select هر دو MUX ذکر شده بیت ۴ ام opcode است چرا که برای دستورات R-format صفر و برای دستورات I-format خواسته شده یک است.

همچنین توجه داریم هر بار pc با یک جمع می‌شود تا در کلاک بعدی دستور بعد از حافظه خوانده شود.



اما برای پیاده سازی واحد کنترلی با توجه به ۶ بیت opcode و همچنین 6 بیت funct عمل می‌کنیم :

aluop ای که alu ساخته شده در تمرین ۳ داشت به شرح زیر است:

- ADD = 0
- SUB = 1
- MUL = 2
- DIV = 3
- AND = 4
- OR = 5
- XOR = 6
- CLO = 7
- CLZ = 8
- SLL = 9
- SRL = 10
- SRA = 11
- ROTR = 12

بنابراین اگر دستور I-format بود (تشخیص آن با چک کردن ۱ بودن بیت چهارم opcode است)چون تنها دستور این نوع addi است واحد کنترلی باید برای alu aluop=0000 (کد جمع قرار دهد) . حال اگر دستور R-format بود باید به ۶ بیت funct توجه شود :

عملیات	funct بیت 6	aluop بیت 4
add	100000	0000
sub	100010	0001
or	100101	0101
and	100100	0100
xor	100110	0110
sll	000100	1001
srl	000110	1010
sra	000111	1011

حال با توجه به جدول زیر بر اساس funct باید ۴ بیت aluop را با استفاده از مدار های ترکیبی مشخص کنیم که به صورت زیر خواهد شد :

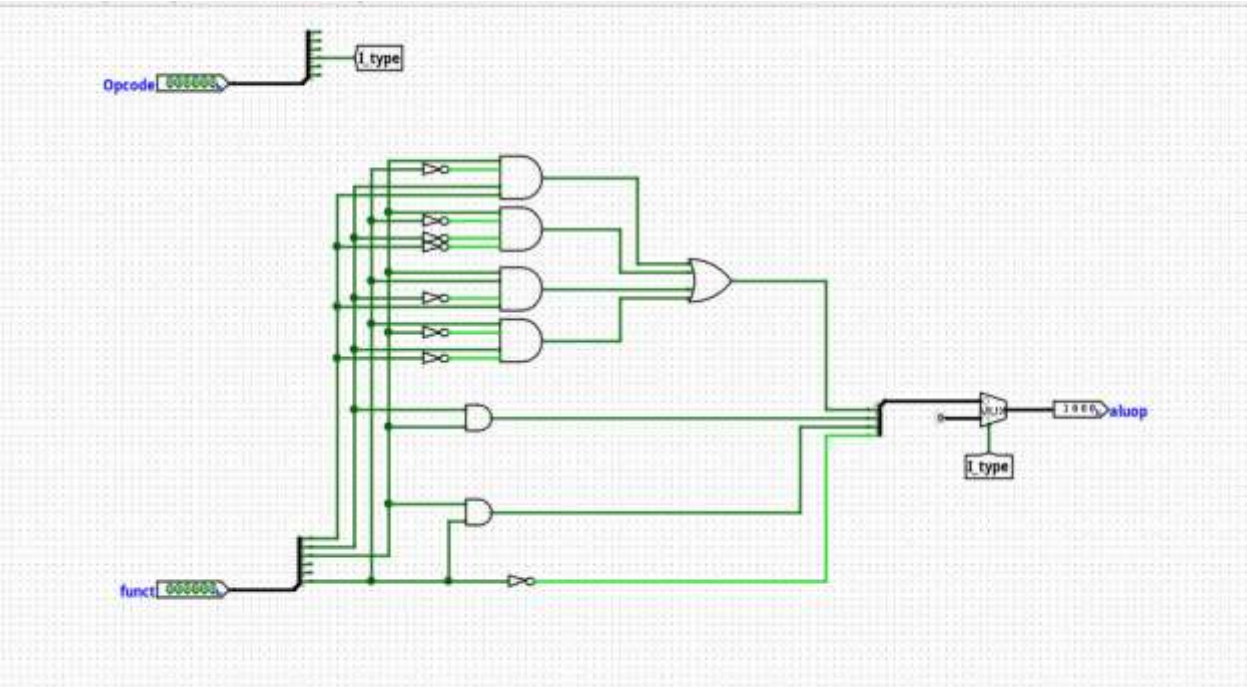
$$\text{aluop}_3 = \overline{funct_5}$$

$$aluop_2 = funct_5 \ \& \ funct_2$$

$$aluop_1 = funct_1 \ \& \ funct_2$$

$$aluop_0 = funct_5 \ \& \ \overline{funct_2} \ \& \ \overline{funct_0} \ \& \ funct_1 + funct_5 \ \& \ funct_2 \ \& \ \overline{funct_1} \ \& \ funct_0 + \overline{funct_5} \ \& \ funct_2 \ \& \ \overline{funct_1} \ \& \ \overline{funct_0} + \overline{funct_5} \ \& \ funct_2 \ \& \ funct_1 \ \& \ funct_0$$

واحد کنترلی را در زیر مشاهده می‌کنیم:



همچنین پاس شدن تست را نیز مشاهده می‌کنیم:

```
Q11p.circ.tmp/main/verilog/circuit/jtag_ram64.v
[main] INFO com.cburch.logisim.fpga.gui.Reporter - Creating HDL file: /home/erfan/logisim_evolution_workspace/
Q11p.circ.tmp/main/verilog/circuit/jtag_ram512.v
[main] INFO com.cburch.logisim.fpga.gui.Reporter - Creating HDL file: /home/erfan/logisim_evolution_workspace/
Q11p.circ.tmp/main/verilog/gates/AND_GATE_4_INPUTS.v
[main] INFO com.cburch.logisim.fpga.gui.Reporter - Creating HDL file: /home/erfan/logisim_evolution_workspace/
Q11p.circ.tmp/main/verilog/gates/OR_GATE_4_INPUTS.v
[main] INFO com.cburch.logisim.fpga.gui.Reporter - Creating HDL file: /home/erfan/logisim_evolution_workspace/
Q11p.circ.tmp/main/verilog/circuit/Control_Unit.v
[main] WARN com.cburch.logisim.fpga.gui.Reporter - Component "Register" in circuit "main" has a gated clock co
nnection!
[main] INFO com.cburch.logisim.fpga.gui.Reporter - Creating HDL file: /home/erfan/logisim_evolution_workspace/
Q11p.circ.tmp/main/verilog/circuit/main.v
[main] INFO com.cburch.logisim.fpga.gui.Reporter - Creating HDL file: /home/erfan/logisim_evolution_workspace/
Q11p.circ.tmp/main/verilog/toplevel/logisimTopLevelShell.v
[main] INFO com.cburch.logisim.fpga.gui.Reporter - Creating script file: /home/erfan/logisim_evolution_workspa
ce/Q11p.circ.tmp/main/scripts/vivadoCreateProject.tcl
[main] INFO com.cburch.logisim.fpga.gui.Reporter - Creating script file: /home/erfan/logisim_evolution_workspa
ce/Q11p.circ.tmp/main/xdc/vivadoConstraints.xdc
[main] INFO com.cburch.logisim.fpga.gui.Reporter - Creating script file: /home/erfan/logisim_evolution_workspa
ce/Q11p.circ.tmp/main/scripts/vivadoGenerateBitStream.tcl
[main] INFO com.cburch.logisim.fpga.gui.Reporter - Creating script file: /home/erfan/logisim_evolution_workspa
ce/Q11p.circ.tmp/main/scripts/vivadoLoadBitStream.tcl

HW4/tb.v /home/erfan/logisim_evolution_workspace/Q11p.circ.tmp
ACCEPTED
        21 /        21
```