به‌نام خدا

تمرین 5 معماری

نوید رئیـــــس‌زاده – محمد بهرامـــــی.

- سوال 1 :

هنگامی که رجیستر فایل ما دارای یک ورودی باشد نیاز است که ابتدا با اضافه کردن یک مالتیپکسر سه به یک سه ورودی rs1, rs2 , rd را به تک ورودی رجیستر فایل متصل کنیم و یک سیگنال دو بیتی کنترل(RegFileInCtrl) را به کنترل یونیت اضافه کنیم.

در هیچ سایکلی همزمان read و write را نیاز نداریم ولی در دستورات R-type و beq ما نیاز داریم که هم rs1 و هم rs2 را بخوانیم که نیاز است در دو سایکل این مورد انجام شود و در دو رجیستر nonarchitectural بعد از رجیستر فایل ذخیره شوند.

چون یک ورودی داریم یه خروجی هم داریم که در سایکل اول نیاز است تا rs1 را بخوانیم و در رجیستر A ذخیره کنیم و در سایکل دوم rs2 را.

برای این منظور از خروجی یک سیم به رجیستر A و یک سیم به B میبریم و در سایکل اول enable رجیستر A را فعال کرده و در سایکل دوم enable رجیستر دوم را که نقیض enable اولی است.نام enable را RegOutEn میگذاریم.

در FSM مان یک استیت اضافه میکنیم که اگر دستور R-type یا beq بود بعد از استیت Decode به آن استیت میرویم تا rs2 را بخوانیم و بعد از آن اگر دستور beq بود به استیت BEQ رفته و اگر دستور R-type بود به ExecuteR میرویم.

در قسمت هایی که نیاز به write کردن هست ، باید کنترل مالتیپلکسر را 10 قرار دهیم.

DataPath:

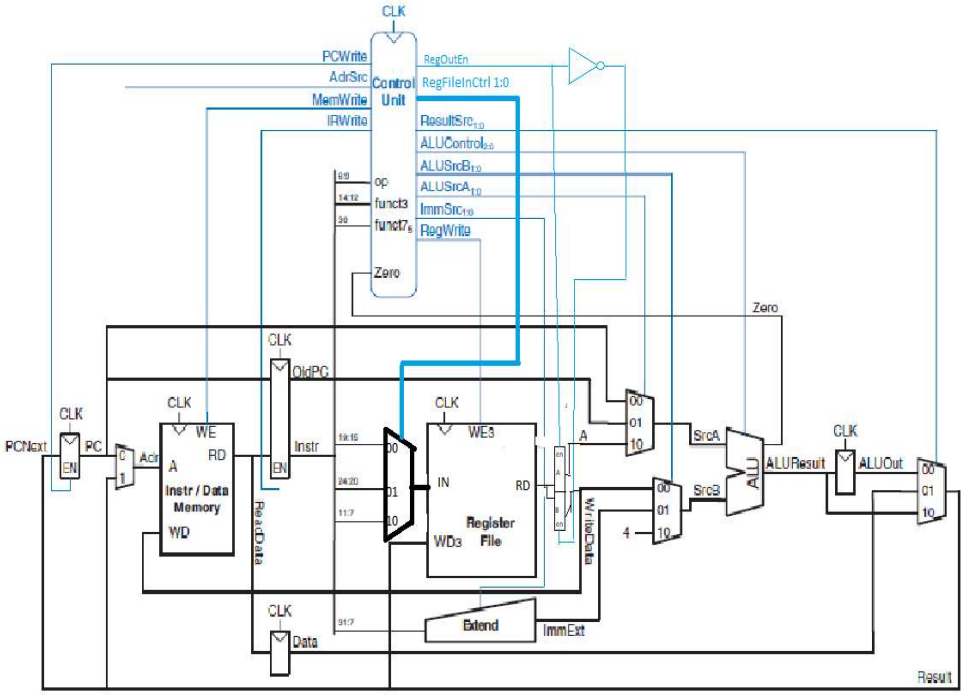
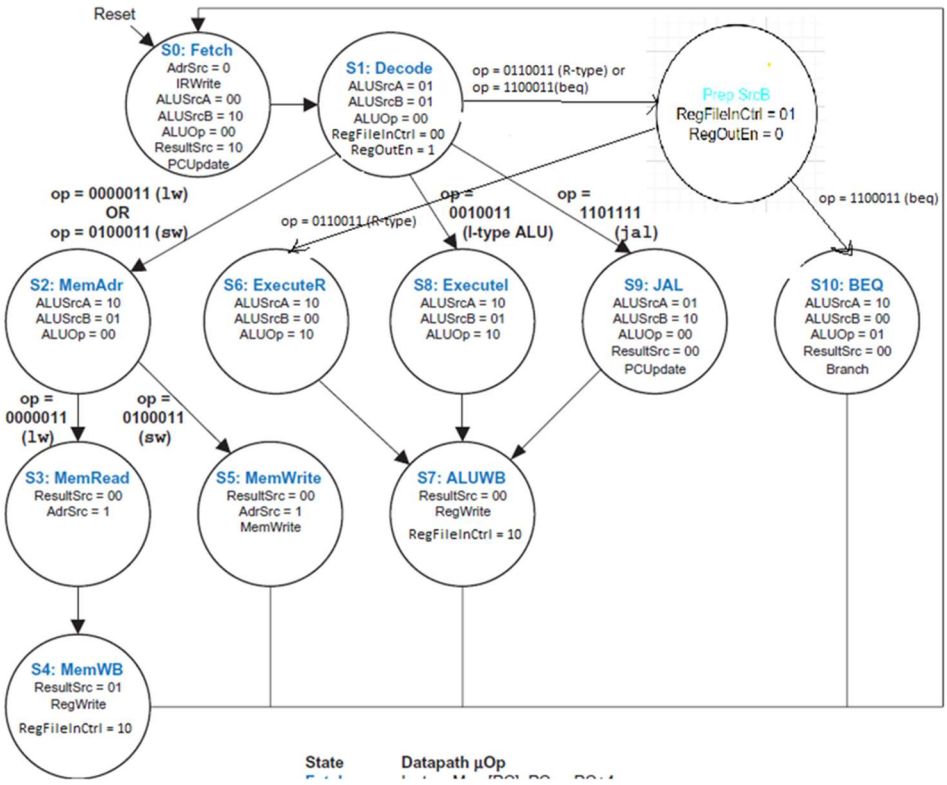


Figure 7.27 Complete multicycle processor

FSM:

نام استیت اضافه شده را PrepB قرار میدهیم و پس از ست کردن دستورات برای خواندن B اگر opcode برای beq بود به استیت BEQ رفته و اگر برابر با R-type بود به استیت ExecuteR میرویم

در دو استیت ALUWB, MemWB باید مقدار RegFileInCtrl را برابر 10 قرار دهیم تا مقداری که باید رایت شود انتخاب شود



پی‌نوشت: تصاویر FSM و Data Path به صورت جداگانه و با نسخه‌ی باکیفیت درر فایل‌بندی موجود است.

- سوال دوم :

ابتدا تعداد اجرای هر خط را می‌شماریم. هر خط بیرون حلقه‌ی loop یکبار اجرا می‌شود و داخل حلقه‌ی for نیز 7 بار اجرا میگردد. البته که شرط حلقه 8 بار اجرا میگردد.

تابع فوق تابع فیبوناچی است و جمع دو عدد قبل از هر عدد را محاسبه میکند و به عنوان جروجی 7امین عضو فیبوناچی را باز میگرداند.

برای محاسبه‌ی تعداد کلاک‌ها، تعداد اجرای هر خط را باید در تعداد کلاک‌های هر instruction ضرب کنیم. تعداد کلاک‌های هر instruction را میتوان بر اساس data path آن یافت.

برای محاسبه‌ی CPI هم تعداد کلاک‌ها را بر تعداد اینستراکشن‌ها تقسیم می‌کنیم.

```
addi  t0 , zero , 0x10010
```

t1 مقدار ۷ را میگیرد و به‌عنوان شرط پایان حلقه

```
addi  t1 , zero , 7
addi  t2 , zero , 0
addi  t3 , zero , 1
addi  t5 , zero , 1
```

loop در نظر می‌گیریم.

```
sb    t2 , 0(t0)
sb    t3 , 1(t0)
```

مقدار t_3، t_5 را برابر با ۱ در نظر می‌گیریم.

```
loop:
```

کد، تابع فیبوناچی را محاسبه می‌کند.

```
bgt   t5 , t1 , end
addi  t5 , t5 , 1
add   t4 , t2 , t3
add   t0 , t0 , t5
sb    t4 , 0(t0)
mv    t2 , t3
mv    t3 , t4
sub   t0 , t0 , t5
j     loop
end:
```

تمامی کدهای خارج loop، هرخط را یکبار اجرا می‌کند. درون حلقه loop، خط اول که شرط پایان حلقه است ۸ بار اجرا می‌شود و سایر خط‌ها درون حلقه ۷ بار.

هر دستور کلاک ۴ و sb و sub و add و addi

j و mv هم ۴ کلاک - bgt دارای ۳ کلاک است.

۸ بار bgt : $۳ \times ۸ = ۲۴$ ، $۷ \times ۴ = ۲۸$ و $۵۶ \times ۴ = ۲۲۴$

دستورهای بیرون حلقه

۲۷۶ تعداد کلاک سایکل

$$CPI = \frac{۲۷۶}{۷۱} = ۳,۸۹$$

سوال سوم:

دستور srai یک نوع دستور I-type است.

0010011 (19)	101	0100000*	I	srai rd, rs1, uimm	shift right arithmetic imm.	rd = rs1 >>> uimm

تنها تغییری که باید اعمال شود تغییر در ALU است تا بتواند دستور srai را اجرا کند.

ششمین دستوری که ALU میتواند اجرا کند shift right arithmetic خواهد بود که به آن کد 100 را اختصاص میدهیم (در گذشته به slt مقدار 101 را اختصاص داده بودیم در صورتی که کد 100 هنوز استفاده نشده بود). حال باید با ALU Decoder Truth Table را متناسب با srai تغییر دهیم.

ALUOp	funct3	$\{op_5, funct7_5\}$	ALUControl	Instruction
00	x	x	000 (add)	lw, sw
01	x	x	001 (subtract)	beq
10	000	00, 01, 10	000 (add)	add
	000	11	001 (subtract)	sub
	010	x	101 (set less than)	slt
	110	x	011 (or)	or
	111	x	010 (and)	and

10	101	01	100	srai

Opb5 = 0 و funct7b5 =1 بودن.

نیازی به تغییر دیتاپث و FSM نیست و فقط ALU باید تغییر میکرد .