به نام خدا

گزارشکار آز single_cycle میکروپروسسور

سید محمد حسن میرشفیعی ۴۰۲۱۰۲۵۵۱

کد وریلاگ تکمیل شده single_cycle_mips.v همراه این فایل فرستاده شده است

در این آز باید بخش control unit از یک پردازنده میپس سینگل سایکل را پیاده سازی کنیم؛ قسمت های خواسته شده برای پیاده سازی در زیر آمده اند:

R format: add, sub, addu, subu, and, or, xor, nor, slt, sltu
I format: addi, addiu, slti, sltiu, andi, ori, xori, lui, beq, bne, lw, sw

دستورات R-format با op صفر در واقع با funct که ۶ بیت lsb از instruction است کنترل میشوند و بقیه دستورات با همان op کنترل می شوند (دلیل اینکه تمامی دستورات داخل فیلد op قرار نگرفته محدودیت بیت برای فرمت I و J است که چون در دستورات R-format چون از ۳۲ بیت ۶ بیت خالی (function) داریم میتوان از آن برای کنترل این دسته از دستورات استفاده کرد). در ادامه به کد مربوطه می پردازیم:

خروجی های اصلی کنترل یونیت شامل موارد زیر هستند:

- PCSrc: برای پرش در دستورات beq و bne
- SZEn: فعال‌سازی Sign Extend
- AluOP: انتخاب عملیات ALU
- ALUSrc: انتخاب بین عملوند دوم Immediate یا رجیستر
- RegDst: انتخاب بین rd/rt برای نوشتن
- MemtoReg: انتخاب منبع نوشتن در رجیستر
- RegWrite: نوشتن در رجیستر
- MemWrite: نوشتن در حافظه

```
6'b000000: begin // R-type
  RegDst    = 1'b1;
  ALUSrc    = 1'b0;
  MemtoReg  = 1'b0;
  RegWrite  = 1'b1;
  MemWrite  = 1'b0;
  SZEn      = 1'bx;
  case(func)
    6'b100000: AluOP = `ADD;
    6'b100001: AluOP = `ADD;
    6'b100010: AluOP = `SUB;
    6'b100011: AluOP = `SUB;
    6'b100100: AluOP = `AND;
    6'b100101: AluOP = `OR;
    6'b100110: AluOP = `XOR;
    6'b100111: AluOP = `NOR;
    6'b101010: AluOP = `SLT;
    6'b101011: AluOP = `SLTU;
    default  : AluOP = 4'bxxxx;
  endcase
end
```

Control Signal	R-type (add, sub, and, or, slt...)
RegDst	1
ALUSrc	0
MemtoReg	0
RegWrite	1
MemWrite	0
SZEn	X
AluOP	Derived
PCSrc	0

مطابق با جدول بالا باید برای این تایپ از دستورات سیگنال کنترلی را اساین کنیم؛ در قسمت AluOP بسته به دستور func خروجی لازم ست شده است. تنها چیزی که مقدار صریح برای آن ست نشده PCSrc است که چون سیگنال از نوع reg نیست لذا اگر به درستی مقداردهی نشود ممکن است باعث جامپ به جای پرت شود که برای این مشکل راه حل را در ادامه خواهیم گفت.

از دستورات R-type تنها یکی را بررسی خواهیم کرد lw (load word):

```
6'b100011: begin // lw
  RegDst    = 1'b0;
  ALUSrc    = 1'b1;
  MemtoReg  = 1'b1;
  RegWrite  = 1'b1;
  MemWrite  = 1'b0;
  SZEn      = 1'b1;
  AluOP     = `ADD;
end
```

Control Signal	lw (Load Word)
RegDst	0
ALUSrc	1
MemtoReg	1
RegWrite	1
MemWrite	0
SZEn	1
AluOP	ADD
PCSrc	0

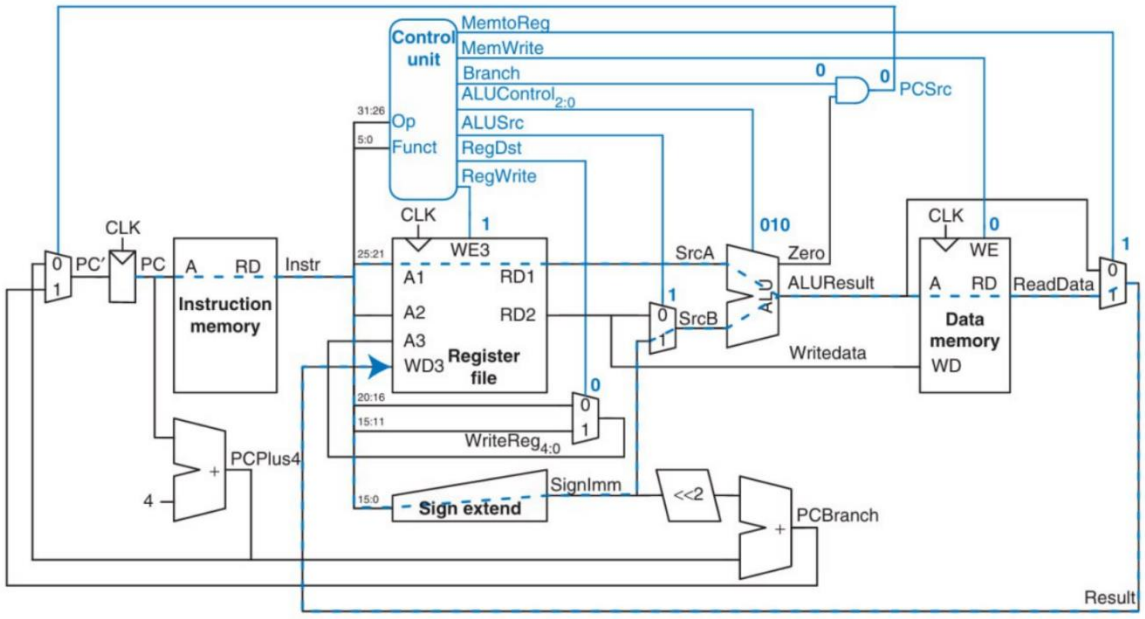

Figure 7.15 Critical path for lw instruction

رجیستر RegDst باید صفر باشد چون:

مقصد (رجیستر rt) در فرمت I-type در بیت‌های [۲۰:۱۶] قرار دارد، و نه [۱۵:۱۱] مانند دستورهای R-type.

رجیستر AluSrc باید یک باشد چون:

یکی از ورودی‌های ALU باید مقدار offset (ثابت) باشد و نه رجیستر دوم. بنابراین از Immediate استفاده می‌شود.

رجیستر MemtoReg باید یک باشد چون:

داده‌ای که از حافظه خوانده شده باید به عنوان خروجی به رجیستر بنویسیم، نه خروجی ALU.

رجیستر RegWrite باید یک باشد چون:

چون داده‌ای که از حافظه می‌خوانیم باید در رجیستر مقصد (rt) نوشته شود، باید این سیگنال فعال باشد.

رجیستر MemWrite باید صفر باشد چون:

این دستور چیزی در حافظه نمی‌نویسد، بلکه از آن می‌خواند. بنابراین نوشتن در حافظه غیرضروری است.

رجیستر SZEn باید یک باشد چون:

در بسیاری از پیاده‌سازی‌ها برای استفاده از Immediate مقدار آن باید sign-extend شود؛ چون ممکن است منفی باشد.

وایر PCSrc باید صفر باشد چون:

چون این دستور جهشی (branch) نیست، نباید PC به مقدار دیگر تغییر یابد. مسیر عادی ادامه دارد.

رجیستر AluOP باید در حالت جمع ADD باشد چون:

برای محاسبه آدرس مؤثر (offset + rs) از جمع استفاده می‌شود ALU .باید عمل جمع انجام دهد.

```
54    // CONTROLLER COMES HERE
55    assign PCSrc = (op == 6'b000100 && AluZero) || (op == 6'b000101 && ~AluZero);
```

برای کنترل کردن بیت کنترلی PCSrc نیز از اساینمنت فوق استفاده می کنیم. که با دستورات beq , bne مقدار PCSrcرا برای جامپ کردن ست می کند.