

بسمه تعالی

درس ساختار کامپیوتر و میکروپروسسور

استاد: دکتر موحدیان عطار

دانشکده مهندسی برق



گزارش فاز دوم پروژه درس (پیاده سازی پردازنده single cycle بر اساس دستور العمل mips)

امیرحسین صفری

۹۷۱۰۱۹۹۴

واحد های سازنده پردازنده :

واحد ControlUnit :

در این واحد با پس از ورودی گرفتن ، سیگنال های op (opcode) ، $funct$ (function) و $zero$ ، سیگنال های کنترلی خروجی را به عنوان ورودی به ماژول $datapath$ می دهیم ، برای این کار ابتدا با استفاده از $opcode$ ، R_type یا I_type بودن دستورات چک شده و سپس با استفاده از $switch$ ، سیگنال های کنترلی مختص هر دستور مشخص می شود (برای دستورات R_type ، فقط خروجی $ALUop$ ، مختص هر دستور و در واقع بر اساس $function$ ورودی تعیین می شوند و سایر سیگنال های کنترلی مثل هم می باشند ، اما در دستورات I_type ، تمام سیگنال های کنترلی مختص هر دستور و بر اساس $opcode$ ، مشخص می شوند)

واحد datapath :

در این واحد ، ابتدا clk و $reset$ را ورودی می گیریم و در ادامه سیگنال های خروجی واحد $ControlUnit$ می باشد به علاوه سیگنال $instr$ (instruction) از واحد $Instruction_mem$ (Instruction memory) و هم چنین سیگنال $readdata$ از واحد $t_DataMemory$ (data memory) را به عنوان ورودی می گیریم .

و هم چنین سیگنال های خروجی $zero$ ، pc ، alu_result ، $writedata$ را برای ورودی دادن به $ControlUnit$ و $Instruction memory$ و $data memory$ را تولید می کنیم .
برای ساخت این واحد از واحد های زیر استفاده شده است : (نام سیگنال ها بر اساس شکل صفحه بعد انتخاب شده اند)

(۱) ماژول $reset_ff$: یک $D_flipflop$ که بر سر راه pc قرار می گیرد تا در صورت $reset$ شدن مدار ، pc برابر با صفر وارد $Instruction memory$ شود .

(۲) ماژول $PCplus4$: برای ساختن pc بعدی (سیگنال $PCplus4$) این سیگنال در صورت اتفاق نیفتادن $branch$ وارد $Instruction memory$ می شود .

(۳) ماژول shift : از این ماژول برای در ۴ ضرب کردن (شیف्ट دادن به تعداد ۲ به چپ) سیگنال SignImm استفاده می شود .

(۴) ماژول adder : با استفاده از این ماژول سیگنال PCPlus4 را با ۴ برابر SignImm جمع کرده و سیگنال PCBranch را می سازد . (این سیگنال در صورتی که branch رخ داده باشد ، وارد Instruction memory می شود)

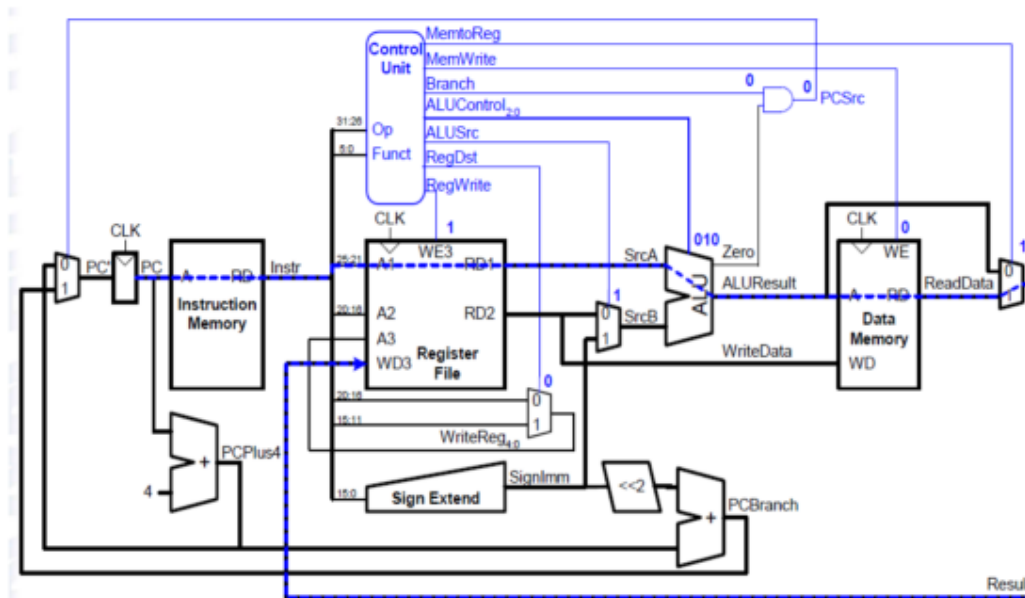
(۵) ماژول MUX : از این ماژول در ۴ مکان مختلف به شرح زیر استفاده می گردد :

(۱) مشخص کردن اینکه ورودی PCBranch یا PCPlus4 وارد Instruction memory شود .

(۲) برای اینکه سیگنال WriteReg را مشخص کند ، در واقع آدرس رجیستری که در آن داده ذخیره می شود را بر اساس اینکه دستور R_type یا I_type می باشد ، مشخص می کند.

(۳) برای مشخص کردن ورودی دوم (SrcB) واحد ALU ، از بین داده ی موجود در رجیستر اگر دستور R_type باشد و یا داده ی immediate اگر دستور I_type باشد .

(۴) و بالاخره Imux ای که مشخص کند که آیا داده ی خروجی data memory به عنوان سیگنال Result مشخص شود و یا خروجی ALU .



۶) ماژول RegisterFile : برای کار با رجیستر های موجود و نوشتن داده و یا خواندن داده از آن.

۷) ماژول signORzeroExtend : برای sign_extend کردن مقدار immediate (برای دستوراتی مثل beq و addi) یا zero_extend کردن آن (برای دستوراتی مثل andi)

۸) ماژول ALU : برای اینکه خروجی مورد نظر را بر اساس نوع دستور و در واقع سیگنال ALUcontrol .

حال با استفاده از دو ماژول ControlUnit و datapath ماژول mips را ساخته و در واقع در این مرحله پردازنده ساخته می شود ، سپس با اتصال کردن Instruction_mem و t_DataMemory به پردازنده ، ارتباط پردازنده نیز با حافظه برقرار می شود .

نتایج و زمان اجرای تست ها

(۱) تست ۱: sort

برای این تست ضمن اینکه تغییراتی جزئی مطابق با نام مازول ها را در فایل SingleCycle_tb اعمال می کنیم ، آن را اجرا کرده و نتیجه مطابق زیر می شود .

```

Console
Time resolution is 1 ps
Simulator is doing circuit initialization process.
Finished circuit initialization process.
ISim>
# run 400us
ffff8a4f ff49a03e ed9232cf ed9232cf ec6c3298 ec6c3298 e6663185 e6663185 dddd8526 dbdb842d b6b6e9be b4b4e947 aac70a4f aaaaec60 a48e7be0 a48e7be0
9c7a4305 9c7a4305 99bd6c60 8bd654a6 8bd654a6 8894b185 878c0526 86b259c7 82846947 826e5d18 826e5d18 81da5ead 81da5ead 8183b298 8081042d
807f69be 802ab2cf 8019fbe0 8008c305 8007d4a6 8002d9c7 8001dd18 8000dead 8000203e 8000203e 7fff8a4f 7fff8a4f 7f49a03e 7f49a03e 6d9232cf 6c6c3298
66663185 5ddd8526 5ddd8526 5bdb842d 5bdb842d 36b6e9be 36b6e9be 34b4e947 34b4e947 2ac70a4f 2ac70a4f 2aaaec60 2aaaec60 248e7be0 1c7a4305 19bd6c60
19bd6c60 0bd654a6 0894b185 0894b185 078c0526 078c0526 06b259c7 02846947 02846947 026e5d18 01da5ead 0183b298 0183b298 0081042d 0081042d 007f69be
007f69be 002ab2cf 002ab2cf 0019fbe0 0019fbe0 0008c305 0008c305 0007d4a6 0007d4a6 0002d9c7 0002d9c7 0001dd18 0001dd18 0000dead 0000dead 0000203e
Stopped at time : 258350 ns(1) : in File "D:\electrical engineering\computer Architecture\project\phase2\CA Project Phase2\SingleCycle_tb.v" Line 53
ISim>

```

Console Compilation Log Breakpoints Find in Files Results Search Results

Sim Time: 258,350,000 ps Ln 53 Col 1 Verilog

همانطور که در تصویر نیز مشخص است ، زمان اجرای کد 258350 ns می باشد .

(۲) تست Fibonacci Sequence

در ابتدا کد mips ای که ۱۵ عدد اول دنباله ی فیبوناچی را محاسبه و در حافظه ذخیره می کند را در فایل fibtest.asm نوشته و سپس با استفاده از نرم افزار Mars ، فایل hex متناظر با آن را با نام fibtest.hex را تولید می کنیم و در ادامه مطابق آن چه که در دستور پروژه گفته شده فایل تست بنچ fib_tb.v را تکمیل کرده و پس از اجرای آن ، خروجی را به صورت زیر مشاهده می کنیم : (هم چنین زمان اجرا 1280 ns می باشد)

```

Console
ISim P.20131013 (signature 0x7708f090)
This is a Full version of ISim.
Time resolution is 1 ps
Simulator is doing circuit initialization process.
Finished circuit initialization process.
ISim>
# run 40us
00000001 00000001 00000002 00000003 00000005 00000008 0000000d 00000015 00000022 00000037 00000059 00000090 000000e9 00000179 00000262 Stopped at time : 1280 ns(1) :
Architecture\project\phase2\CA Project Phase2\fib_tb.v" Line 57
ISim>

```

Sim Time: 1,280,000 ps Ln 57 Col 1 Verilog