باسمه تعالی

امتحان پایان‌ترم از درس VHDL، بخش یک – امتحان در منزل

هدف از انجام این تکلیف کمک به درک عملکرد یک اسمبلر، معماری یک ریزپردازنده تک سیکلی، معماری یک ریزپردازنده پایپ‌لاین و طراحی ریزپردازنده‌ها توسط زبان توصیف سخت‌افزاری VHDL می‌باشد. در این تکلیف، در نخست از شما خواسته شده تا یک اسمبلر برای ریزپردازنده ساده‌ای به اسم Miniature بنویسید. ویژگی‌های این ریزپردازنده به قرار زیر است:

الف) این پردازنده یک ماشین ۳۲ بیتی است (هر کلمه آن ۳۲ بیت است).

ب) Miniature دارای ۱۶ رجیستر بوده که هر کدام ۳۲ بیت دارد (R0 – R15) و رجیستر R0 همیشه دارای مقدار صفر می‌باشد.

ج) هر واحد آدرس دهی این ماشین یک بایت می‌باشد و چون هر دستورالعمل نیز یک چهاربایت است، PC+4 به دستورالعمل بعدی در سری دستورهای برنامه اشاره دارد.

د) Miniature دارای ۱۰۲۴ بایت و یا ۲۵۶ کلمه حافظه است

ه) این ریزپردازنده سه نوع دستورالعمل و ۱۵ دستور دارد که فرمت آنها در زیر آمده است:

R-type: Instructions: ***add***, ***sub***, ***slt***, ***or***, and ***nand***

```
bits 31-28 unused all zero
bits 27-24 opcode
bits 23-20 rs (source register)
bits 19-16 rt (target register)
bits 15-12 rd (destination register)
bits 11-0  unused (all zero)
```

I-type: Instructions: ***addi***, ***ori***, ***slti***, ***lui***, ***lw***, ***sw***, ***beq*** and ***jalr***

```
bits 31-28 unused all zero
bits 27-24 opcode
bits 23-20 rs (source register)
bits 19-16 rt (target register)
bits 15-0  offset
```

J-type: Instructions: ***j*** and ***halt***

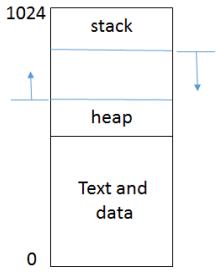
```
bits 31-28 unused all zero
bits 27-24 opcode
bits 23-16 unused and they should be all zero
bits 15-0  target address
```

دقت بفرمایید که فیلد rs از دستور lui و فیلد offset از دستورالعمل jalr هر دو صفر می‌باشند.

جزئیات دستورالعمل‌های این ریزپردازنده در جدول ۱ و شمای حافظه Miniature در شکل ۱ آمده است.

جدول ۱: دستورالعمل‌های ریزپردازنده Miniature

mnemonic				opcode	Description
add	rd	rs	rt	0000	rd <- rs + rt, PC <- PC + 4, if overflow then ov = '1'
sub	rd	rs	rt	0001	rd <- rs - rt, PC <- PC + 4, if overflow then ov = '1'
slt	rd	rs	rt	0010	if rs < tr then rd <- 1
					else rd <- 0, PC <- PC + 4
or	rd	rs	rt	0011	rd <- rs or rt, PC <- PC + 4
nand	rd	rs	rt	0100	rd <- rs nand rt, PC <- PC + 4
addi	rt	rs	offset	0101	rt <- rs + SE(offset)
ori	rt	rs	offset	0110	rt <- rs or ZE(offset)
slti	rt	rs	offset	0111	if rs <- offset then rt <- 1
					else rt <- 0, PC <- PC + 4
lui	rt	offset		1000	rt(31-16) <- offset, rt(15-0) <- "0…00"
lw	rt	rs	offset	1001	rt <- Mem(rs + offset), PC <- PC + 4
sw	rt	rs	offset	1010	Mem(rs + offset) <- rt, PC <- PC + 4
beq	rt	rs	offset	1011	if (rs == rt) zero = '1', PC <- PC + 4 + SE(offset)
					else zero ='0', PC <- PC + 4
jalr	rt	rs		1100	rt <- PC + 4, PC <- rs
j	offset			1101	PC <= ZE(offset)
noop				1110	PC <- PC + 4
halt				1111	PC <- PC + 4, then halt the machine



شکل ۱: شمای حافظه ریزپردازنده Miniature

قسمت اول این تکلیف دو بخش دارد و این دو بخش از اهمیت یکسانی برخوردار هستند. بخش اول طراحی و پیاده سازی اسمبلر است و بخش دوم تست و راستی آزمایی اسمبلر نوشته شده.

۱. طراحی و پیاده‌سازی یک اسمبلر به زبان C: برای این ماشین اسمبلری طراحی و به زبان C پیاده سازی کنید که نام دستورالعمل‌های این ریزپردازنده که به زبان اسمبلی هستند را به معادل باینری آنها تبدیل کند. درضمن این اسمبلر باید برچسب‌هایی را نیز که در هنگام نوشتن برنامه به زبان اسمبلی از آنها استفاده می‌شود، به معادل آدرس آنها تبدیل کند.

خروجی این اسمبلر سری از دستورالعمل‌های ۳۲ بیتی است که فرمت آن به صورت داده شده زیر می‌باشد:

label<white>instruction<white>field0,field1,field2<white>#comments

دقت بفرمایید که white یک یا چند فاصله و یا tab می‌باشد. در توضیح باید اشاره کرد که اولین فیلد همان برچسب است که حداکثر از ۶ کاراکتر تشکیل شده است و اگرچه باید با یک حرف انگلیسی شروع شود، اما می‌تواند اعداد را نیز شامل شود. با اینکه فاصله پس از برچسب لازم است، اما وجود خود برچسب منطقی است که اختیاری باشد. بعد از فاصله ضروری قید شده، دستورالعمل‌های نشان داده شده در جدول یک، ظاهر می‌شوند. نهایتا هر دستورالعمل فیلدهای مختص خود را داراست و برای نمایش رجیسترها کافی است تنها عدد آنها در دستور اسمبلی قید شود.

تعداد فیلدهای یک دستورالعمل به نوع آن بستگی دارد و فیلدهای که مورد استفاده قرار نمی گیرند باید صرفنظر شوند. به عنوان مثال دستورالعملهای نوع R دارای ۳ فیلد بوده که فیلد اول rd، فیلد دوم rs و فیلد سوم rt می باشد. آدرسهای سمبولیک به برچسب ها اشاره دارند. برای دستورالعملهای lw و sw، اسمبلر باید آدرس برچسب آنها را محاسبه کرده و با یک رجیستر پایه غیر صفر که در این صورت خانه های یک آرایه را مورد اندیس قرار می‌دهد، جمع کند. در صورتی که در دستورهای lw و sw رجیستر پایه صفر باشد، آدرس محاسبه شده برچسب در این دستورها جایگزین می شود. برای دستورالعمل beq، اسمبلر نیاز است تا برچسب را به مقدار عددی offset تبدیل کند (که برای آن انشعاب ضروری است). شایان ذکر است که پس از آخرین فیلد، فاصله قرار می گیرد و هر توضیحی که به صورت اختیاری ظاهر می شود، باید با علامت # همراه شود. فیلد توضیح با پایان یافتن هرخط پایان می یابد.

به انضمام دستورالعملهای Miniature، یک برنامه اسمبلی ممکن است شامل Directive نیز باشد. تنها directiveهای این ماشین ".fill" و ".space" می باشند که اولی عددی را در حافظه قرار می دهد و دومی به تعداد داده شده خانه های حافظه ذخیره می کند که البته مقدار آنها صفر شده است.

در مثال زیر "start .fill" مقدار ۲ را در آدرس ۸ حافظه قرار می دهد. در ضمن ناگفته نماند که برچسب StAddr مقدار ۸ می گیرد.

اسمبلری که طراحی می کنید لازم است کد اسمبلی را دوبار مرور کند. در (اصطلاحاً) scan اول، اسمبلر معادل عددی هر برچسب را محاسبه می کند و هر دو برچسب و معادل عددی آنرا در جدولی به نام Symbol Table ذخیره می کند. در مرور دوم اسمبلر کد اسمبلی را به زبان ماشین ترجمه می کند و در حین ترجمه از جدول symbol table استفاده می کند تا معادل عددی هر برچسب را نیز جایگزین کند.

در ادامه یک برنامه اسمبلی آمده است که کد ماشین آن نیز داده شده است. خواهشمند است با دقت کافی این برنامه و معادل کد ماشین آنرا مطالعه کنید تا در طراحی و پیاده سازی اسمبلر برای Miniature با مشکل کمتری مواجه شوید.

```
        lw      1 0 five
        lw      2 1 4
start   add     1 1 2
        beq     0 1 done
        j       start
done    halt
five    .fill   5
neg1    .fill   -1
StAdd   .fill   start
```

```
(address 0)  0x09010018
(address 4)  0x09120004
(address 8)  0x00121000
(address 12) 0x0b100014
(address 16) 0x0d000008
(address 20) 0x0f000000
(address 24) 0x00000005
(address 28) 0xffffffff
(address 32) 0x00000008
```

دقت بفرمایید که گرچه در کد ماشین فوق آدرسها برای بهتر تفهیم شدن ترجمه اسمبلر قید شده اند، اما آنچه نیاز است اسمبلر شما به صورت خروجی تولید کند، به صورت زیر می باشد.

(دقت بفرمایید که خط فاصله های فقط برای خوانا بودن هستند) 0000 1001 0000 0001 0000 0000 0001 1000

...

۲. اجرای اسمبلر: اسمبلر را چنان بنویسید که دو آرگمان در خط دستور (Command Line)، به صورت زیر، دریافت.

assemble program.as program.mc

همانطوری که مشخص است assemble فایل قابل اجرای اسمبلر می باشد، برنامه اسمبلی شما در program.as ذخیره شده است و نهایتاً اسمبلر کد ترجمه شده به زبان ماشین را در program.mc ذخیره می کند. لازم به توضیح است که دقیقا مانند مثال فوق، هر خط کد ماشین در program.mc، یک عدد باینری (معادل باینری کد اسمبلی) می باشد. هر خروجی دیگری مانند کد منظور شده برای debugging که توسط شما در اسمبلر نوشته می شود، باید در standard output چاپ شود.

۳. خطاهای قابل تشخیص با اسمبلر: اسمبلر شما باید قادر باشد خطاهای زیر را تشخیص بدهد:

الف) استفاده از برچسب تعریف نشده

ب) برچسب های که تعریف شده اند و بیش از یک بار استفاده شده اند

ج) offsetی که در ۱۶ بیت نمی گنجد

د) opcode تعریف نشده

اسمبلر در صورت بروز خطا با exit(1) اجرای عمل اسمبلی را متوقف کرده و در صورتی که هیچ خطای را تشخیص ندهد با exit(0) عمل اسمبلی را به پایان می رساند. دقت بفرمایید که اسمبلر نباید خطاهای در حین اجرا مانند «انشعاب به آدرس ۱-» و یا «حلقه نامتناهی» را تشخیص بدهد.

٤. راستی آزمایی: برای راستی آزمایی اسمبلری که نوشته اید، لازم است مجموعه ای از برنامه های اسمبلی نیز تهیه کنید. این مجموعه برنامه، برنامه‌های تقریبا کوتاهی هستند که به عنوان ورودی اسمبلر به منظور آزمایش بکار می روند. در کنار برنامه اسمبلی، لازم است که این مجموعه را نیز تحویل دهید. هر برنامه اسمبلی تست باید حداقل ۱۰ خط بوده و به ۵ نمونه تست نیاز است. سعی کنید هر برنامه اسمبلی را طوری بنویسید که قمستهای متفاوتی از اسمبلر را تست کند.

قسمت دوم این تکلیف طراحی تک سیکلی این ریزپردازنده است. در این طراحی از جمع‌کننده carry lookahead استفاده شود. در آغاز هر کدام از مولفه‌های این ریزپردازنده را جداگانه طراحی و به زبان VHDL کد رفتاری آنهارا بنویسید. به عنوان مثال از رجیسترفایلی که در کلاس طراحی شد استفاده کنید. البته نیاز است تا جمع‌کننده‌ای را که طراحی کردید کمی تغییر دهید. در آخر مولفه‌ها را بصورت ساختاری ترکیب کرده و ریزپردازنده را کامل کنید. کدخود را توسط مدل سیم شبیه‌سازی کرده و از صحت رفتاری آن مطلع شوید. کدهای اسمبل شده قسمت اول را توسط این ریزپردازنده اجرا کنید. زمان اجرا آنها را برحسب تعداد سیکل محاسبه کنید.

قسمت سوم – ریزپردازنده پایپ لاین: ریزپردازنده‌ای که در قسمت دوم شبیه‌سازی کردید را اکنون به یک ریزپردازنده پایپ لاین تبدیل کرده کد آنرا تکمیل و شبیه‌سازی کنید. کد خود را به یک برد اف پی جی ای نگاشت کنید. این ریزپردازنده پایپ لاین هیچگونه هازارد داده و یا هازارد کنترلی را نه تنها به صورت سخت‌افزاری برطرف نمی‌کند بلکه تشخیص هم نمی‌دهد. بنابراین وظیفه نرم‌افزار است تا با اضافه کردن noop در بین خطوط برنامه که به هم وابستگی دارند از بروز خطاهای داده و کنترلی (data and control hazard) جلوگیری کند.

در آخر دو برنامه بنویسید که هر کدام دارای data hazard و control hazard به طور جداگانه باشد. به صورت دستی تعداد موردنیاز noop را به کد اضافه کنید تا منجر به محاسبات صحیح بشود. دو قطعه کد خود را توسط اسمبلری که نوشته اید، اسمبل کرده و با ریزپردازنده خود اجرا کنید. در یکی از این برنامه‌ها حتما یک حلقه با حداقل ۱۰۰ بار تکرار وجود داشته باشد.

قسمت چهارم – پایپ‌لاین با تشخیص هازارد: در این فاز از پروژه از شما خواسته شده تا طراحی خود را چنان غنی کنید که تشخیص هازارد داده و هازارد کنترلی با سخت‌افزار بوده و به مقدار نیاز توسط سخت‌افزار حباب به پایپ‌لاین تزریق شود. اصولا از نظر علمی تفاوت چندانی بین کارایی ریزپردازنده قسمت چهارم و قسمت سوم نخواهد بود، گرچه ریزپردازنده قسمت سوم به مقدار ناچیزی بهتر از ریزپردازنده قسمت چهارم عمل می‌کند. برای قسمت هازارد کنترل، از تکنیک پیش بینی و آنهم پیش بینی "always not taken" استفاده کنید.

قسمت پنجم – پایپ‌لاین با تشخیص و برطرف کردن هازارد: در این فاز قرار براین است که ریزپردازنده شما از تکنیک "data forwarding" برای برطرف کردن هازارد داده هم در گامهای "exe – exe" و هم در گامهای "mem – exe" لذت ببرد. در ضمن برای برطرف کردن هازارد کنترل نیز از تکنیک پیش‌بینی «مانند دفعه گذشته» استفاده شود. دوقطعه کدی که برای قسمت چهارم نوشتید را توسط ریزپردازنده این قسمت نیز اجرا کنید و میزان تسریع در عملکرد را نسبت به قسمت چهارم گزارش کنید.

موفق باشید.