باسمه تعالی

هدف از انجام این پروژه کمک به درک عملکرد یک اسمبلر و چگونگی ساختار آن است. در این پروژه 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 کلمه می‌باشد و چون هر دستورالعمل نیز یک کلمه است، PC+1 به دستورالعمل بعدی در سری دستورهای برنامه اشاره دارد.

د) Miniature دارای ۶۵۵۳۶ 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***

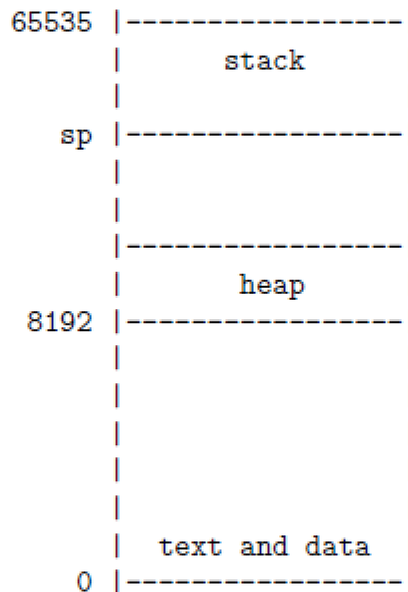
```
bits 31-28 unused all zero
bits 27-24 opcode
bits 23-16 unused and they should be all zero
bits 15-0  target address
```

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 + 1
sub  $rd,$rs,$rt        0001     $rd <- $rs - $rt, PC <- PC + 1
slt  $rd,$rs,$rt        0010     if ($rs < $rt)  $rd <- 1
                                 otherwise       $rd <- 0
                                 , PC <- PC + 1
or   $rd,$rs,$rt        0011     $rd <- $rs | $rt, PC <- PC + 1
nand $rd,$rs,$rt        0100     $rd <- $rs & $rt, PC <- PC + 1
addi $rt,$rs,imm        0101     $rt <- $rs + SE(imm), PC <- PC +1
slti $rt,$rs,imm        0110     if ($rs < SE(imm) $rt <- 1
                                 otherwise         $rt <- 0
                                 , PC <- PC + 1
ori  $rt,$rs,imm        0111     $rt <- $rs | ZE(imm), PC <- PC + 1
lui  $rt,imm            1000     $rt <- imm << 16, PC <- PC + 1
lw   $rt,$rs,offset     1001     $rt <- Mem($rs + SE(offset)), PC <- PC + 1
sw   $rt,$rs,offset     1010     Mem($rs + SE(offset)) <- $rt, PC <- PC + 1
beq  $rt,$rs,offset     1011     if ($rs == $rt) PC <- PC + 1 + SE(offset)
                                 else PC <- PC + 1
jalr $rt,$rs            1100     $rt <- PC + 1, PC <- $rs
j    offset             1101     PC <- ZE(offset)
halt                    1110     PC <- PC + 1, then halt the machine
```



شکل ۱: شمای حافظه ریزپردازنده Miniature

این پروژه ۲ قسمت دارد که این دو از اهمیت یکسانی برخوردار هستند. قسمت اول طراحی و پیاده سازی اسمبلر است و قسمت دوم تست و راستی آزمایی اسمبلر نوشته شده.

۱. طراحی و پیاده‌سازی یک اسمبلر به زبان Python: برای این ماشین اسمبلری طراحی و به زبان Python 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 ".fill start" 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   #  load reg1 with 5 (symbolic address)
        lw      2,1,2      #  load reg2 with -1 (numeric address)
start   add     1,1,2      #  decrement reg1
        beq     0,1,done   #  goto end of program when reg1==0
        j       start      #  go back to the beginning of the loop
done    halt               #  end of program
five    .fill   5
neg1    .fill   -1
stAddr  .fill   start      #  will contain the address of start (2)
```

```
(address 0): 151060486 (hex 0x09010006)
(address 1): 152174594 (hex 0x09120002)
(address 2): 1183744 (hex 0x0121000)
(address 3): 184614913 (hex 0x0b010001)
(address 4): 218103810 (hex 0x0d000002)
(address 5): 234881024 (hex 0x0e000000)
(address 6): 5 (hex 0x5)
(address 7): -1 (hex 0xffffffff)
(address 8): 2 (hex 0x2)
```

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.

```
151060486
152174594
1183744
184614913
218103810
234881024
5
-1
2
```

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 صحیح (معادل ده دهی – به جای 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، برنامه های 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.