به نام خدا

امتحان نهایی ساختار و زبان کامپیوتر ۱۲۶-۴۰

۱۸/۱۰/۱۳۹۱

مدت: دو ساعت

۱- چهار ویژگی متمایز کننده پردازنده MIPS را نسبت به پنتیوم از نظر ISA (معماری مجموعه دستورالعمل) برشمارید.

۲- در پردازنده ARM تعداد زیادی از دستورات (مثل MOV r1,r2) می توانند به طور شرطی اجرا شوند. یعنی برحسب یک پرچم مثل N، C ... اجرا شوند یا نشوند. به نظر شما کدگذاری دستورات ARM چه ویژگی متمایزی نسبت به آنچه تاکنون آموخته اید دارد؟ ترجیحاً با مثال و یا کمی سخن بگویید.

۳- چند قطعه کد داده شده است که معادل آنرا بنویسید:

Code1:

```
xchg eax, ebx
xchg ebx, ecx
xchg ecx, eax
```

Equivalent Instruction:

Code2

```
mov ecx, 0
loop loop1
```

Equivalent Instruction:

(برچسب loop1 در برنامه وجود دارد)

code3

```
shl eax,1
pushfd
jnc next
add eax, 1
next:
popfd
```

Code 4

```
push eax
sub eax, ebx
pop eax
```

۴- با فرض می خواهیم عبارت:

$$x = x + y + z - 9$$

و معجود بودن متغیرهای x، y، z و 9 در به ترتیب ثبات های S_1، S_2، S_3 و S_4، یک برنامه اسمبلی MIPS برای محاسبه فوق بنویسید.

۵- فرض کنید دستوراتی همچون loopz, loopnz, loope, loopne مجاز نیستند (یا وجود ندارند) با چند دستور ساده معادل دستور loopnz target را بیابید و سپس آنرا بنویسید.

۶- این برنامه را تحلیل کنید و مشخص نمایید چه می‌کند

```
TITLE problem
; This program ....

INCLUDE Irvine32.inc
.data
     string1BYTE "This problem is worth 25 points!", 0
.code
main PROC
     mov esi, OFFSET string1
     call proc1

     mov edx, OFFSET string1
     call WriteString
     exit
main ENDP

proc1 PROC
     again:
          cmp BYTE PTR [esi], 0
          jz done
          cmp BYTE PTR [esi], 'a'
          jb next
          cmp BYTE PTR [esi], 'z'
          ja next
          sub BYTE PTR [esi], 20h
     next:
          inc esi
          jmp again
     done:
          ret
proc1 ENDP
END main
```

۷- دسترسی به حافظه به صورت مستقیم (DMA) از کدام دستگاه جانبی (I/O) مستلزم چه تعامل و تبادل اطلاعات یا فرمان‌هایی به/از کدام پردازنده، حافظه و I/O است. ترجیحاً نمودار بکشید.

۸- توضیح دهید اگر حافظه واقعی فقط 16 MByte و حافظه منطقی (مجازی) 1 GByte باشد، مکانیزم نگاشت حافظه مجازی به واقعی با جدول صفحه‌ای و سایر ملزومات به چه شکل باید باشد.

۸-۲- آیا در این ساختار امکان segmentation وجود دارد؟ اگر بلی چگونه و اگر نه چرا؟

۹- تفاوت وقفه نرم‌افزاری Int x با فراخوانی زیربرنامه چیست؟

۱۰ - مقدار آرایه list1 را در هر بخش از برنامه زیر تعیین و توجیه کنید.

```
TITLE Unknown Program
; The program ....

INCLUDE Irvine32.inc

.data
list1 BYTE 10 DUP(?)

.code
main PROC

            mov esi, OFFSET list1
            mov ecx, LENGTHOF list1
            mov al, 0
target1:    mov [esi], al
            inc esi
            inc al
            loop target1

      ; Determine contents of list1 here
      ;


            mov edi, esi
            dec edi
            mov esi, OFFSET list1
            mov ecx, LENGTHOF list1
target2:    mov al, [esi]
            mov bl, [edi]
            mov [esi], bl
            mov [edi], al
            inc esi
            dec edi
            loop target2

      ; Determine contents of list1 here



            exit
main ENDP
END main
```

۱۱ - به طور خیلی خلاصه تفاوت object code و ایمیج و DLL را شرح دهید.

موفق باشید.

به نام خدا

۱۸ تیر ۱۳۹۱

تصحیح امتحان پایان ترم ساختار و زبان کامپیوتر ۱۲۶-۰۴

۲۶

۱ - چهار ویژگی متمایزکننده MIPS نسبت به پنتیوم:

- تعداد بیشتر ثبات‌ها (۳۲ در مقابل ۸)
- طول دستورات ثابت (۳۲ بیتی) در مقابل طول متفاوت در پنتیوم
- حالت آدرس‌دهی پشته مستقیم به جای مستقیم و در پشته مستقیم MIPS ۲۶ بیت آدرس داخل دستورالعمل دوبیت به چپ جابجا شده با ۴ بیت بالای PC جهت ۳۲ بیت آدرس مطلق یا مستقیم را درست می‌کند)
- دستورات ۳ آدرسه مثل (add $t1, $t2, $s6)

۲

ویژگی‌های برتر: - دستورات پرش خود مقایسه را انجام داده پرش را براساس آن انجام می‌دهند مثل beq $s3, $t1, ...
- ثبات صفر ثابت صفر را تأمین می‌کند.

۲ - در ARM چهار بیت مربوط به اجرای شرطی به دستورات اضافه شده است که یا اجرای دستور را لغو می‌کند (مثل NOP) یا اینکه مثل یک دستور واقعی اجرا می‌شود. بنابراین با این ۴ بیت شرطی می‌توان همه دستورات را شرطی اجرا نمود و نیازی به استفاده از دستورات انشعاب شرطی نیست. معمولاً چنین تمهیدی برای تسریع در اجرای پایپلاین و پرهیز از اتلاف وقت در انشعاب و فالی کردن پایپلاین پرارزش می‌اندیشند.

۲

۳ -

Code 1:
```
xchg eax, ebx
xchg ebx, ecx   <=>   xchg ebx, ecx
xchg ecx, eax
```

Code 2:
```
mov ecx, 0   <=>   jmp loop1
loop loop1
```

Code 3:
```
shl eax, 1
pushfd
jnc next
add eax, 1
next: popfd
```
; ذخیره کل ۳۲ بیت پرچم‌ها
<=> rol eax, 1
(اگر بیت سمت چپ eax یک باشد، با افتادن آن به CF جابجا شده، که یک صفر در ورودی دارد، عملاً یک به راست AX وارد شده)

۴

دستور loop قبل از شمارش cx را صفر آزمایی می‌کند و چون مخالف آن در دستور قبلی صفر شده است پس: loop1 پرش می‌کند.

Code 4:
```
push eax   <=> cmp eax, ebx
sub eax, ebx
pop eax
```

ص ۱

S_1 S_2 S_3 S_4

$x = x + y + z - 9$

۴ -

```
add   $S1, $S1, $S2    ; x ← x+y
add   $S1, $S1, $S3    ; x ← x+z
sub   $S1, $S1, $S4    ; x ← x - 9
```

۲

۵ - دستور loop n z Target به روش RISC کار خود (= حلقه زدن) را تکرار می‌کند: یکی اینکه cx صفر نشده باشد و دوم اینکه پرچم z برپا نشده باشد یعنی: شرط پرش $(CX \neq 0) \wedge (Z = 0)$

Target است. برای پیاده‌سازی معادل آن کافی است بنویسیم:

۲

زیرا "loop" محتوای cx را چک می‌کند و قبل از آن لازم شرط z=1 چک شده است.

```
    Jz next
    loop target
next:
```

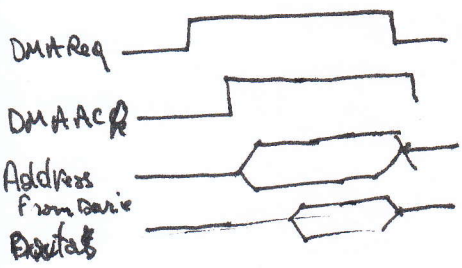
۶ - برنامه حروف کوچک را به حروف بزرگ تبدیل می‌کند زیرا در جدول ASCII حروف کوچک به اندازه 20h بزرگتر از حروف بزرگ هستند مثلاً 'a'=61, 'A'=41

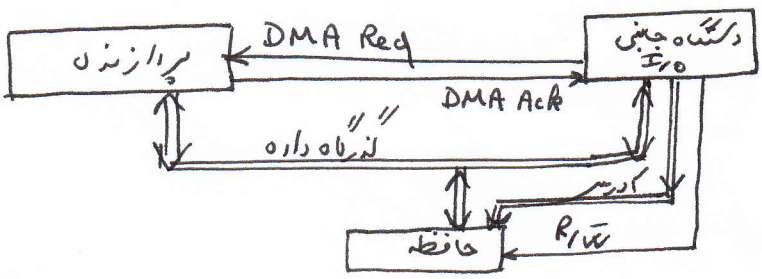
۲

مقایسه‌ها: با صفر برای تشخیص انتهای رشته
با کوچکتر 'a' جهت عدم چک کاراکترهای خاص
و بزرگتر از 'z'

می‌باشد.

۷ -





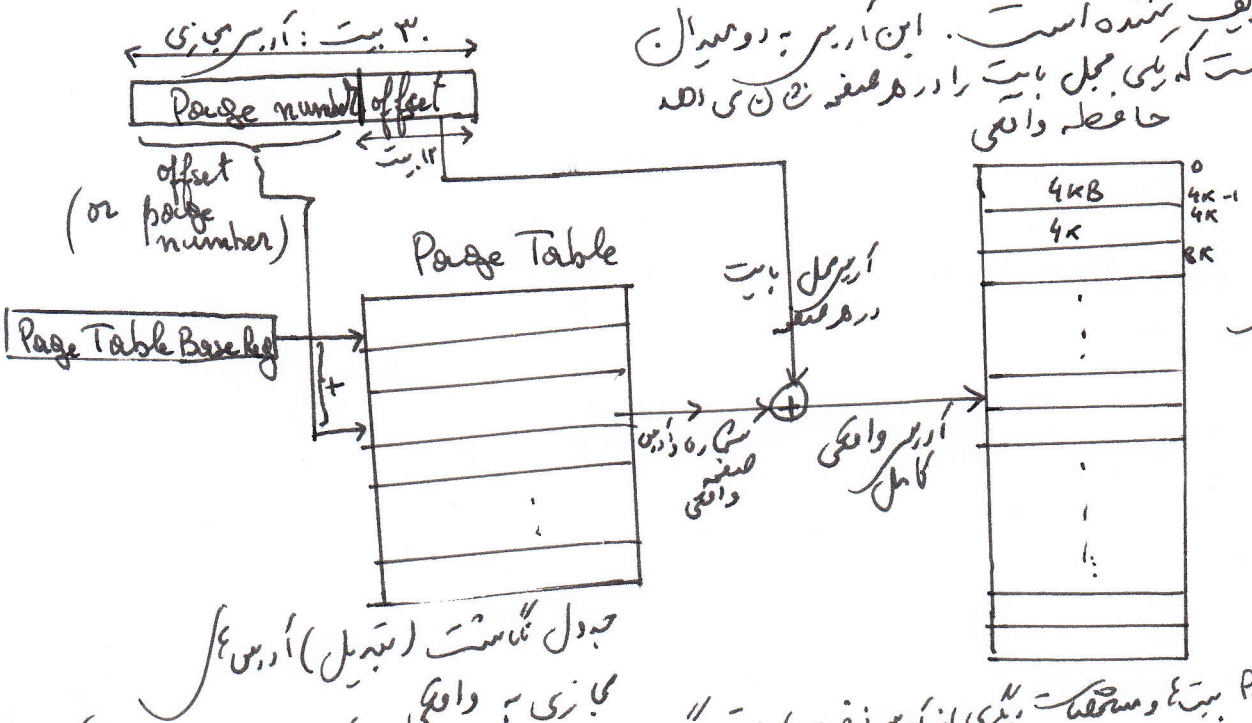
۲

۱ - دستگاه جانبی درخواست DMA را به پردازنده می‌دهد.

۲ - پردازنده (پس از اتمام کار جاری) درخواست دستگاه را با اعلام DMA Ack تصدیق می‌کند و گذرگاه‌های داده و آدرس و کنترلی خود را رها می‌کند (یعنی به حالت امپدانس بالا می‌رود)

۳ - دستگاه جانبی خطوط آدرس حافظه را فعال نموده به سراغ خانه حافظه برای عمل خواندن و نوشتن (R/W) از طریق گذرگاه داده می‌رود. پس از اتمام کارش DMA Req خود را برمی‌چیند تا پردازنده کار خود را با گذرگاه‌ها پی گیرد.

۸-۱ اگر فرض کنیم هر صفحه حافظه (Page frame) برابر 4KByte است بنابراین در حافظه واقعی حداکثر $\frac{2^{24}}{2^{12}} = 2^{12} = 4096$ صفحه وجود دارد. آدرس مجازی ۳۰ بیتی است زیرا 1GByte تعریف شده است. این آدرس به دو میدان تقسیم شده است که یکی مجل بایت را در هر صفحه تعیین می‌کند و دیگری شماره صفحه را که با آدرس پایه جدول صفحه جمع شده و آدرس واقعی را می‌دهد.



جدول نگاشت (تبدیل) آدرس‌های مجازی به واقعی

توجه: در Page table بیت‌ها و مشخصات دیگری از آدرس ذخیره می‌شود تا اگر داده در حافظه وجود نداشت از کجا بیاوریم یا چگونه جایگزین کنیم.

۸-۲ بله می‌توان segmentation را نیز به مکانیزم فوق اضافه نمود. کافی است به جای آنکه آدرس خطی فوق را مستقیم در حافظه عرضه کرد آنرا با بیت (های) segment (یعنی آنچه در 80x86 خوانده می‌شود) ترکیب کرد و سپس آدرس واقعی را بدست آورد.

۹- وقفه Int x تفاوت ماهوی با فراخوانی (call) زیربرنامه ندارد بلکه در تعداد بایت‌های دستورالعمل و تعداد کارهای اضافه‌ای که انجام می‌دهد متفاوت است. اولاً: Int x دو بایت دارد (به جز Breakpoint: Int 3 که یک بایت دارد) حال آنکه call در حالت near، ۳ و در حالت far ۵ بایت دارد. بنابراین فراخوانی از طریق Int x کوتاه‌تر (اقتصادی‌تر) است، البته به شرطی که آدرس محل برش را بیت پیش در جدول برای وقفه در آدرس 4x x قرار داده باشیم. ثانیاً دستور Int x ابتدا ثبات وضعیت را در پشته ذخیره می‌کند، بعد بیت‌های T و I را صفر کرده، سپس CS را در پشته قرار می‌دهد، بعد مقدار جدید CS را از جدول برای وقفه می‌خواند، بعد IP را در پشته ذخیره کرده، مقدار جدید IP را از برای وقفه استخراج نموده، به این آدرس جدید CS:IP خوانده شده از جدول برای وقفه برش می‌کند. حال آنکه در call نزدیک (Near) فقط IP و در call Far CS و IP در پشته ذخیره می‌شود. لذا Int x به Call Far بیشتر شبیه است ولی با آن تفاوت‌هایی طول دستور و تعداد عملیات فوق را دارد.

۱۰_

```
list1   Byte  DUP(?)
        mov   esi, offset list1
        mov   ecx, length of list1
        mov   al, 0
target1: mov  [esi], al
        inc   esi
        inc   al
        loop  target1
```

این حلقه ده بار (طول آرایه list1) اجرا می‌شود و مقدار al را که هر بار یکی اضافه می‌شود در خانه‌های متوالی list1 قرار می‌دهد. پس داریم:

list1 = [0, 1, 2, 3 ——— 9]
list1[0] ... list1[9]

پس از خروج از حلقه esi به list1[10] اشاره می‌کند.

```
        mov   edi, esi      ; edi → list1[9]  (اشاره می‌کند)
        dec   edi
        mov   esi, offset list1   ; esi → list1[0]
        mov   ecx, length of list1 ; cx = 10
target2: mov  al, [esi]
        mov   bl, [edi]
        mov   [esi], bl
        mov   [edi], al
        inc   esi
        dec   edi
        loop  target2
```

این حلقه ۱۰ بار اجرا می‌شود و هر بار درایه i را با n-i جابجا می‌کند یعنی:

list1[0] ⟷ list1[9]
list1[1] ⟷ list1[8]
list1[2] ⟷ list1[7]
list1[3] ⟷ list1[6]
list1[4] ⟷ list1[5]
list1[5] ⟷ list1[4]
list1[6] ⟷ list1[3]
list1[7] ⟷ list1[2]
list1[8] ⟷ list1[1]
list1[9] ⟷ list1[0]

پس چون دوبار هر عنصر با نظیر خود (i ⟷ n-i) جابجا شده است، عملاً جابجایی رخ نداده و list1 همان آرایه فوق است.

list1 = [0, 1, 2, ... 9]

۱۱_ object code = کد باینری قابل فهم برای CPU که توسط برنامه‌ای به نام اسمبلر از کد ASM به دست می‌آید.

درایور: برنامه‌ای که به سیستم‌عامل می‌گوید چگونه با یک سخت‌افزار خاص (مثل چاپگر، کارت شبکه، کارت گرافیک) صحبت کند.

DLL: Dynamic Link Library — فایلی که شامل توابع و کدهایی است که برنامه‌ها می‌توانند هنگام اجرا به آن لینک شوند و از آن استفاده کنند.