به نام خدا

امتحان میان‌ترم ساختار و زبان کامپیوتر (۴۰۱۲۶) و تصحیح ۱۵ آذر ۱۴۰۲

(مدت: ۲ ساعت)

۱- توضیح دهید چرا به جای پیاده‌سازی ۴ مدار متفاوت برای ۴ عمل اصلی می‌توان فقط با مدار جمع‌کننده و یک مبدل مناسب (که نام می‌برید و توضیح می‌دهید) همه چهار عمل اصلی را ساده‌تر پیاده‌سازی سخت‌افزاری کرد. کدام ماشین‌های مکانیکی از این ایده و چگونه بهره بردند؟

پاسخ: از آنجا که منفی یک عدد را می‌توان با مکمل‌گیری آن به دست آورد پس عمل تفریق معادل جمع با مکمل عدد است. ضرب (عدد صحیح) هم معادل جمع‌های متوالی (مثلاً 5x3=5+5+5) است و تقسیم هم معادل تفریق‌های متوالی (با لحاظ باقیمانده نهایی). پس به کمک جمع و مکمل‌گیر می‌توان بقیه عمل‌های اصلی حسابی را انجام داد. ماشین پاسکال جمع و تفریق را پیاده کرده بود و بر اساس آن، ماشین لایبنیز ضرب و تقسیم را پیاده کرد. البته در ماشین پاسکال، جمع زدن دو عدد با به جلو چرخاندن چرخ‌دنده‌های ماشین به تعداد مورد نظر برای جمع انجام می‌شد و تفریق کردن با به عقب چرخاندن چرخ‌دنده‌ها. هر گاه نیز رقم نقلی (Carry) (یا قرضی (Borrow)) رخ می‌داد، چرخ‌دنده سمت چپ یکی به جلو (یا عقب) رانده می‌شد.

۲- مقادیر *x* و *y* را در معادله زیر بیابید:

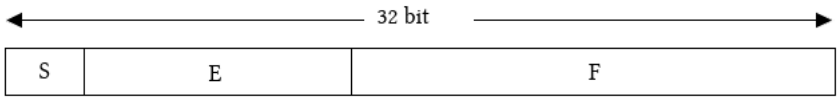
$$(63)_8 + (x)_4 = (E5)_{16}$$
$$(24)_y + (15)_y = (42)_y$$

پاسخ: برای محاسبه مقادیر x و y اعداد را اول به مبنای ۱۰ می‌بریم و بعد محاسبات لازم را در معادلات به‌دست‌آمده انجام می‌دهیم:

$$(63)_8 + (x)_4 = (E5)_{16} \quad \rightarrow \quad (6\times 8+3)_{10} + (x)_4 = (14\times 16+5)_{10}$$
$$\rightarrow \quad (x)_4 = (178)_{10} \quad 178 = 2\times 4^3 + 3\times 4^2 + 0\times 4^1 + 2 \quad \rightarrow \quad x = 2302$$

$(24)_y + (15)_y = (42)_y$ ➔ 2y+4 +y+5= 4y+2 ➔ 3y+9=4y+2 ➔ y=7

۳- فرض کنید قالب اعداد ممیز شناور ۳۲ بیتی به شکل زیر است (البته طول بخش‌های مختلف آن را شما باید تعیین کنید)



$$(-1)^S \times (1.F) \times 2^{E-Bias}$$

اگر عدد $-53\frac{7}{9}$ در این سیستم به صورت $CB5C71C7_{hex}$ نمایش داده شود، مقدار Bias چقدر است؟

پاسخ: اول معادل دودویی (باینری) عدد نوشته شده به Hex را می‌نویسیم:

$CB5C71C7_{hex}=1100101101011100011100011100011_2$

برای تعیین میادین مختلف این نمایش چنین استدلال می‌کنیم:

بیت سمت چپ (۱) نشانه علامت منفی عدد است که جدا می‌کنیم. مقدار $53\frac{7}{9}$ از یک بخش صحیح و یک بخش اعشاری تشکیل شده است که بخش صحیح آن برابر است با: $53=(110101)_2=(1.10101)_2 \times 2^5$.

از آنجا که اعداد مطابق صورت مسئله و فرمول داده شده هنجارشده (Normalized) با بیت ۱ ضمنی فرض شده‌اند، یعنی به صورت 1.F ضرب‌در توانی از ۲ نمایش داده می‌شوند، پس $53\frac{7}{9}$ نیز به صورت 1.F نمایش داده می‌شود که در آن F قطعاً باید با 10101 شروع شود. پس با یک جستجوی ساده در عدد باینری فوق از سمت چپ، قالب یا الگوی 10101 را به‌راحتی می‌یابیم و بدین ترتیب محل شروع میدان F را پیدا می‌کنیم (قسمت خاکستری). پس داریم:

$1.10101\ldots \times 2^5 = 53\frac{7}{9}$

پس در قسمت چپ F مقدار جابجاشده، نما را می‌یابیم: $\text{Biased Exponent}=100101_2$ که برابر 37 است. لذا:

$\text{Bias}=37-5=32=2^5$

بدین ترتیب میدان علامت ۱ بیت، میدان نما ۶ بیت و میدان بخش جزئی (Fractional) یا همان اعشاری ۲۵ بیت دارد که در شکل، هر کدام با رنگ متفاوت نشان داده شده است.

۴- در یک پردازنده، قالب دستورات ۳ آدرسه و ۱۱۲ کد عملیات (Opcode) مختلف تعریف شده است. حافظه ۴ مگابایت ظرفیت دارد و آدرس‌دهی به یک بایت آن از طریق یک ثبات صورت می‌گیرد که در قالب دستورالعمل مشخص شده است (کلاً ۳۲ ثبات داریم). طول دستورات و حداقل تعداد بیت‌های ثبات‌ها چقدر است و چرا؟

OPCODE	REG1	REG2	REG3

پاسخ: با توجه به اندازه حافظه که ۴ مگابایت است (یعنی $۲^{۲۲}$ بایت) پس آدرس حافظه ۲۲ بیتی است و از آن‌جایی که مراجعه به حافظه و عرضه آدرس بدان از طریق ثبات صورت می‌گیرد، پس اندازه یا طول هر ثبات باید دست‌کم ۲۲ بیت باشد.
۳۲ ثبات داریم پس به ۵ بیت برای شناسایی هر کدام نیاز است. در دستورالعمل هم ۳ آدرس (عملوند) تعیین شده است که می‌شود ۱۵ بیت. از سوی دیگر ۱۱۲ کد عملیات داریم که با ۷ بیت قابل شناسایی یا توصیف است. لذا طول میدان Opcode دست‌کم ۷ بیت است که با ۱۵ بیت قبلی می‌شود جمعاً ۲۲ بیت برای حداقل طول دستورالعمل‌ها. بدین سان با ثبات‌های حداقل ۲۲ بیتی (چه برای ثبات‌های عادی و چه برای ثبات‌های PC و IR) می‌توان به سراغ حافظه رفت و دستورات و داده‌ها را واکشی و جستجو کرد یا داده‌ها را در آن نوشت.

۵- فرض کنید دستور Push و Pop نداریم ولی مد آدرس دهی Post-increment و Pre-decrement و دستور Mov دو عملونده و ثبات اشاره‌گر پشته به نام A7 داریم. نشان دهید که چگونه می‌توان دستور Push D1 و Pop D4 را با امکانات این پردازنده پیاده‌سازی یا معادل‌سازی کنیم.

پاسخ: در مد آدرس دهی Post-increment اول سراغ حافظه می‌رویم و انتقال را انجام داده، سپس ثبات آدرس (یا اشاره‌گر) را افزایش می‌دهیم. بالعکس در مد آدرس دهی Pre-decrement ، اول مقدار ثبات اشاره‌گر را کاهش داده، سپس به آدرس حافظه مراجعه کرده، انتقال را انجام می‌دهیم. با فرض انتقال از سمت راست به چپ در دستور، اشاره A7 به راس پشته (Stack) و نوشتن روی خانه بالای آن هنگام Push همراه با کاهش آدرس و خواندن از راس پشته هنگام Pop، دستورات Push و Pop به‌راحتی پیاده‌سازی می‌شود: (علامت - قبل از پرانتز به معنای کاهش پیشین و علامت + بعد از پرانتز، به معنای افزایش پسین است)

Push D1: Mov -(A7), D1
Pop D4: Mov D4, (A7)+

۶- مشخص کنید پس از اجرای برنامه زیر، محتوای انباشتگر و خانه‌های حافظه چه خواهد بود؟ (در همان مبنای اعداد شکل)

(a) LdI 20 (Load Immediate)

(b) LdA 20

(c) $LdInd$ 20 (Load Indirect)

(d) $LdInd$ 30

Adr	Mem
60	50
50	70
40	60
30	50
20	40
10	20

پاسخ:

LdI 20: Acc ← 20 ➔ Acc=20
LdA: Acc ← Mem[20] ➔ Acc=40
LdInd 20: Acc ← Mem[Mem[20]]=Mem[40] ➔ Acc=60
LdInd 30: Acc ← Mem[Mem[30]]=Mem[50] ➔ Acc=70

مقدار خانه‌های حافظه تغییر نمی‌کند چون فقط با دستورات Load به داخل انباشتگر روبرو بودیم.

۷- کامپیوتری داری حافظه‌ای با کلمات ۱۶ بیتی (۲ بایتی) و ۳۲ ثبات است . فرض کنید معماری مجموعه دستورالعمل (ISA) این کامپیوتر طوری باشد که همه دستورات آن داری دو عملوند (Operand) و هر دستور از یک کد عملیات (Opcode) و دو میدان (Field) برای مشخص کردن ثبات‌ها تشکیل شده است.

الف) طول میدان‌های مورد نیاز برای هر دستورالعمل چند بیت است؟

ب) این کامپیوتر حداکثر چند دستورالعمل را می‌تواند اجرا کند؟

ج) نوع معماری این کامپیوتر (CISC/RISC) چیست؟ چرا؟

پاسخ:

الف) $(Opcode, RegAddress1, RegAddress2) = (6,5,5)bits$

ب) $2^6 = 64\ Instructions$

ج) بیشتر به معماری RISC نزدیک است زیرا: طول دستورات ثابت است، تعداد دستورات نسبتاً کم و تعداد ثبات‌ها نسبتاً زیاد است.

۸- این عبارت را با زبان اسمبلی فرضی یا ابداعی و معنی‌دار خود برای ۴ ماشین: بدون آدرس (پشته‌ای)، تک آدرسی (مبتنی بر انباشتگر: Accumulator)، دو آدرسی و سه آدرسی به کمک ثبات‌های دلخواه برنامه‌نویسی و سپس با هم مقایسه یا نقد کنید:

X=(A/B - C)/(Dx6+E)

پاسخ:

ماشین بدون آدرس	ماشین مبتنی بر انباشتگر (یک آدرسه)	ماشین دو آدرسه	ماشین سه ادرسه
Push A	Ld A	Div A,B	Div X,A,B
Push B	Div B	Sub A,C	Sub X,X,C
Div	Sub C	Mul D,6	Mul Y,D,6
Push C	St temp1	Add D,E	Add Y,Y,E
Sub	Ld D	Div A,D	Div X,X,Y
Push D	Muli 6	Mov X,A	
Push 6	Add E		
Mul	St temp2		
Push E	Ld temp1		
Add	Div temp2		
Div	St X		
Pop X	(ld U: load from memory address U to Acc; St V: Store to memory Address V)		

طول برنامه ماشین پشته‌ای یا بدون آدرس (البته به استثناء دستورات Push و Pop که نیازمند عملوند هستند) بیش از بقیه است و هر چه تعداد عملوندهای دستورات بیشتر می‌شود برنامه ظاهراً کوتاهتر می‌شود (یعنی طبعاً تعداد دستورات کمتری لازم دارد) ولی دستورات پر عملوند طول بیتی بیشتری دارند. بنابراین با یقین نمی‌شود گفت کدام برنامه از نظر تعداد بایت مصرفی کوچکتر است. ماشین‌های پشته‌ای عموماً در کمک‌پردازنده‌های حسابی استفاده شده‌اند.

۹- برای برنامه زیر و بر اساس مدهای آدرس‌دهی هر دستور که نام می‌برید، آدرس موثر (Effective address) را بیابید.

DS = 5010 H , BX = 1105H , SI = 00F7H , CX = 01E2H , DI = 0B11H
OFFSET ARRAY = 0149H

```
MOV [0184H],AX
MOV CX,[BX + 4]
MOV [BX + 2×SI],DX
MOV ARRAY[CX+DI],DX
```

پاسخ:

MOV [0184H],AX : DS × 10H + DISP = 50100H + 0184H = 50284H آدرس‌دهی مستقیم (یا مطلق)

MOV CX,[BX + 4]: DS × 10H + BX + 4 = 50100H + 1105H + 4 = 51205H: آدرس‌دهی نسبی به ثبات

MOV [BX + 2×SI],DX : DS × 10H + BX + 2×SI = 50100H + 1105H + 2 × 0007H = 51213H: آدرس‌دهی نمایه (ایندکس) مقیاس شده

MOV ARRAY[CX+DI],DX : DS × 10H + ARRAY + CX + DI = 50100H + 0149H+ 01E2H + 0B11H = 50F3CH: آدرس‌دهی پایه و نسبی به ثبات

۱۰- در کد زیر، مقدار مربوط به محل یا فاصله پرش را در محل نقطه‌چین محاسبه و تعیین کنید.

```
0000 33DB                  XOR BX,BX
0002 B8 0001       START:  MOV AX,1
0005 03 C3                 ADD AX,BX
0007 E9 …Offset1…….        JMP NEXT

                <skipped memory locations>

0200 8B D8         NEXT:   MOV BX,AX
0202 E9 …Offset2…….        JMP START
```

پاسخ: سه نوع پرش داریم: Short، Near و Far در اولی، فواصل پرش به طور نسبی محاسبه می‌شود و بین ۱۲۸- و ۱۲۷+ است. دومی نیز نسبی است و بازه پرش بین ۳۲۷۳۶- و ۳۲۷۳۵+ =۱- $۲^{۱۵}$ است. سومی از نوع آدرس‌دهی مطلق است و دو بایت برای آدرس و دو بایت نیز برای CS پیش‌بینی می‌شود. در مثال بالا، وقتی پردازنده شروع به اجرای دستور نخست JMP NEXT در آدرس 0007 می‌کند، شمارنده برنامه خود (PC) را یا باید به مقدار 0009 و یا 000A بسته به فاصله پرش (Offset1) مقداردهی کند. از آن‌جا که مقصد پرش بدون شرط (Jump) نخست در آدرس 0200 است و فاصله‌اش از ۱۲۷ بیشتر است پس یک پرش از نوع Near داریم و در نتیجه، کد دستورالعمل Jump سه بایتی و مقدار PC برابر 000A خواهد بود (E9 و ۲ بایت فاصله پرش به جلو). این فاصله این‌گونه حساب می‌شود:

Offset1=0200-000A=01F6 -> E9 F6 01

مشابهاً برای پرش دوم، که در آدرس 0007 واقع است، مقدار PC برابر 0205 است و باید به عقب (یعنی آدرس 0002) پرش کند که بدین سان محاسبه می‌شود:

Offset2=0002-0205= -(0203)Hex=FDFD -> E9 FD FD

۱۱- حدس بزنید این برنامه چکار می‌کند و چرا یک سری Push قبل و بعد از Call دارد (یا Pop) و یا چرا MOV BP,SP می‌کند و بعد به کمک BP و نه SP سراغ پشته می‌رود؟ آن را کمی شرح دهید.

```
0000 B8 001E              MOV   AX,30
0003 BB 0028              MOV   BX,40
0006 50                   PUSH  AX
0007 53                   PUSH  BX
0008 E8 0066              CALL  ADDM


0071                ADDM  PROC  NEAR
0071 55                   PUSH  BP
0072 8B EC                MOV   BP,SP
0074 8B 46 04             MOV   AX,[BP+4]
0077 03 46 06             ADD   AX,[BP+6]
007A 5D                   POP   BP
007B C2 0004              RET   4
007E                ADDM  ENDP
```

پاسخ: برنامه بالا یک زیربرنامه جمع دو عدد را صدا می‌زند و برای این کار اول به کمک دو ثبات AX و BX مقادیر را آماده و سپس آن‌ها را برای استفاده آتی توسط زیربرنامه در پشته ذخیره و سپس زیربرنامه ADDM را فراخوانی (Call) می‌کند. در این زیربرنامه، مقدار BP موقتاً در پشته ذخیره می‌شود، چون قرار است به عنوان اشاره‌گر و به جای SP به سراغ داده‌های پشته که یا قبل از فراخوانی در آن نوشته شده‌اند یا در حین محاسبات مورد نیاز زیربرنامه است برود. به واقع، SP را فقط برای Push و Pop نگه می‌دارند و BP را برای دسترسی به میانوندها (Arguments) و داده‌ها و یا اشاره‌گرهای مبادله شده بین برنامه فراخواننده (Caller) و زیربرنامه فراخوانده شده (Callee).

دستور Call یا از نوع Near، یعنی با آدرس نسبی پرش ۱۶ بیتی و طول کد کلاً ۳ بایتی است یا از نوع Far، که چهار بایت برای آدرس پرش (مطلق) و یک بایت هم برای Opcode. پردازنده قبل از اجرای این دستور، آدرس مراجعت را در پشته ذخیره می‌کند که طبعاً برای حالت Near می‌شود ۲ بایت.

شکل پشته بعد از فراخواندن زیربرنامه ADDM و قبل و بعد از اجرای دستور Push BP و Mov BP,SP زیربرنامه بدین شکل است:

SP→	Return address= 000B
SP+2→	40
SP+4→	30

بعد از صدا زدن زیربرنامه ADDM و قبل از اجرای دستور Push BP

SP=BP→	Old BP
BP+2→	000B
BP+4→	40
BP+6→	30

بعد از صدازدن زیربرنامه ADDM و بعد از اجرای دستور Push BP و Mov BP,SP

مقدار جابجایی (Displacement یا Offset) کدشده در آدرس 0009 نیز به‌درستی برابر 0066 یعنی تفاضل آدرس محل پرش به زیربرنامه: 0071 و آدرس بعد از دستور Call یعنی 000B است که برابر 0066=000B-0071 می‌باشد.

زیربرنامه اول SP را در BP کپی کرده و بعد به سراغ [BP+4]، یعنی مقدار قبلی کپی شده در BX (عدد ۴۰) و سپس سراغ [BP+6]، یعنی مقدار قبلی کپی شده در AX (همان عدد ۳۰)، برای جمع زدن رفته است. لذا وقتی از زیربرنامه برمی‌گردد مقدار AX برابر ۷۰ (حاصل‌جمع داده‌های «پاس‌شده» به زیر برنامه)) خواهد بود. (رج. توضیحات بیشتر در صفحه ۲۱۲ فصل ۶ کتاب: The Intel microprocessors, B. B. Brey در CW). Ret 4 نیز از زیربرنامه برمی‌گردد، یعنی آدرس مراجعت را از پشته برداشته داخل PC می‌گذارد. مقدار 4 هم در این دستور نشانگر این است که ۴ خانه پشته را از داده‌های بلامصرف (۴۰ و ۳۰) میانوندها آزاد کن: (SP← SP+4).

۱۲- چرا نمایش ممیز شناور با وجود خطای مطلق نسبتاً زیاد برای اعداد بزرگ و یا نداشتن فاصله ثابت بین اعداد متوالی (در اکتاوهای مختلف...) به عنوان یک نمایش علمی معتبر این قدر استفاده می‌شود؟

نمایش ممیز شناور دقت نسبی ثابتی دارد در نتیجه، چه اعداد کوچک و چه بزرگ باشند، خطای نسبی یکسان کمّی‌سازی (Quantization) در نمایش اعدادی که معادل ماشینی ندارند رخ می‌دهد. برای اعدادی که معادل ماشینی دارند طبعاً خطای نمایش رخ نمی‌دهد. از سوی دیگر، نمایش ممیز شناور با توزیع اعداد در جهان واقعی همخوانی دارد (مطابق قانون Benford لگاریتمیک است، یعنی تعداد اعدادی که با رقم d شروع می‌شوند متناسب با log(1+1/d) است). بنابراین باکی نیست از اینکه اعداد بسیار بزرگ از هم فاصله زیادی داشته باشند و در بازه اعداد نزدیک صفر اتفاقاً چه بهتر که فاصله اعداد متوالی ماشینی خیلی کم باشد تا بتوانیم اعداد بسیار کوچک را از هم تمیز دهیم. بنابراین با تعداد بیت ثابت، نمایش ممیز شناور خود را با مقیاس مورد استفاده (اکتاو) و اندازه مورد نیاز برای نمایش عدد تطبیق می‌دهد تا از همه بیت‌ها به نحو احسن استفاده کند در حالی که در نمایش ممیز ثابت این گونه نیست و وقتی محل ممیز ثابت شد، دقت (مطلق) نمایش نیز ثابت می‌ماند و این برای اعداد کوچک الزاماً کافی یا مناسب نیست.