به نام خدا

تصحیح امتحان میان‌ترم ساختار و زبان کامپیوتر (۴۰۱۲۶) ۱ خرداد ۱۴۰۲

(مدت: ۲ ساعت) (۳۵ نمره)

۱- توضیح دهید چطور ماشین لایبنیتز ضرب و تقسیم را بر اساس جمع/تفریق انجام می‌داد؟ بر این اساس، چرا چهار "عمل اصلی" را می‌توان فقط با عمل جمع و مکمل‌گیری انجام داد؟ (۳ نمره)

ضرب A در یک عدد n را می‌توان معادل جمع‌های متوالی A با خودش به تعداد n بار و تقسیم X بر d را معادل تفریق‌های متوالی d از X (با احتساب یک باقیمانده R) مشابه یک تقسیم پیمانه‌ای دانست.

نمایش مکمل ریشه یک عدد X (یا منفی آن) در ریشه r و با n رقم برابر است با: $C_r(X)= r^n-X$. بنابراین وقتی با n رقم می‌توان اعداد مثبت و منفی را نشان داد، کافی است تفریق را معادل جمع با منفی یک عدد تلقی کنیم. در نتیجه، فقط به جمع‌کننده و مکمل‌گیر نیاز خواهیم داشت و ضرب و تقسیم نیز که به ترتیب معادل جمع‌ها تفریق-های متوالی است به این دو عمل پایه، یعنی جمع و مکمل‌گیر، خلاصه می‌شوند.

۲- نقش یگان کنترل و واحد اجرایی (مسیر داده) را در اجرای هر مرحله از الگوریتم فون‌نیومن مشخص کنید. (۴ نمره)

الگوریتم فون‌نیومن:

مرحله	کار انجام شده	عنوان انگلیسی	نقش یگان کنترل	نقش واحد اجرایی
۱	واکشی دستورالعمل از محل مورد اشاره شمارنده برنامه (PC):	Fetch instruction	خواندن از حافظه دستورالعمل	-
۲	بروزرسانی PC	PC update	افزودن یک واحد (معمولاً ۲، ۴ یا مقدار متغیر) به مقدار PC بر اساس طول دستور خوانده شده	-
۳	ترجمه دستور	Decode instruction	تشخیص دستور و فعال کردن بخش‌های مرتبط واحد اجرایی	-
۴	خواندن عملوندهای مورد نیاز دستور	Read operand(s)	صدور فرمان خواندن به بانک ثبات‌ها یا حافظه و راه‌گزین‌های گذرگاه	مراجعه به ثبات‌ها یا حافظه داده و خواندن داده‌های مورد نیاز دستور و انتقال داده‌ها به واحد حسابی-منطقی (ALU)
۵	اجرای دستور	Execute	صدور فرمان لازم به بخش حسابی-منطقی یا اجرایی	تولید نتیجه توسط واحد محاسباتی یا اجرایی
۶	پس‌نویسی نتایج در ثبات/حافظه مقصد	Writeback	صدور فرمان نوشتن به ثبات یا حافظه مقصد	ذخیره نتیجه در مقصد
۷	بازگشت به مرحله ۱	Goto1	بازنشانی فرمان‌ها برای اجرای مجدد این حلقه	-

۳- چرا مدل اجرایی رایانه‌های امروزی را برنامه ذخیره شده (Stored Program) می‌نامند و با طرز کار انسان چه تفاوتی دارد؟ اولین ماشینی که این الگو را پیاده کرد کدام بود؟ (۲ نمره)

چون برنامه را در حافظه ذخیره و دستورات را به ترتیب از آن خوانده، ترجمه و اجرا می‌کند. ما الزاماً دستورات را در آدرس‌های متوالی ذخیره و داده‌های را نیز از آدرس‌های مشخصی جستجو نمی‌کنیم.
ماشین تحلیلی چارلز بابیج این مدل را برای بار اول پیاده کرد.

۴- اعداد علامت‌دار را با پنج رقم دهدهی نمایش داده‌ایم (یعنی هم مثبت و منفی و عدد صفر) ولی از نشانه یا بیت علامت استفاده نمی‌کنیم.

۴-۱ بازه نمایش این اعداد را در سیستم نمایش مکمل ۱۰ مشخص کنید. (۱ نمره)

[-۵۰۰۰۰،۴۹۹۹۹]

۴-۲ نمایش عدد منفی ۲۳۴۵۶ چیست؟ (۱ نمره)

۷۶۵۴۴ برابر منفی ۲۳۴۵۶ است. کافی است برای راستی‌آزمایی، ۲۳۴۵۶ را با ۷۶۵۴۴ جمع بزنیم و نتیجه را با ۵ رقم مشاهده کنیم که می‌شود صفر.

۴-۳ عدد مثبت ۴۵۰۶۹ را (فقط با جمع و مکمل‌گیری) منهای مقدار ۳۶۷۲۹ بکنید. (۱ نمره)

۶۳۲۷۱=۳۶۷۲۹- در نتیجه: ۴۵۰۶۹+۶۳۲۷۱= ۴۵۰۶۹-۳۶۷۲۹ که می‌شود: ۸۳۴۰

۴-۴ سوال قبل را در سیستم نمایش مکمل ۱۶ پاسخ دهید. (۱ نمره)

در مبنای ۱۶، منفی $(۳۶۷۲۹)_{۱۶}$ یا مکمل ۱۶ این عدد برابر است با: $(۳۶۷۲۹)_{۱۶}$ -$۱۶^۵$ یعنی: (C98D7).
حاصل تفریق می‌شود: $(0E940)_{۱۶}$=$(C98D7)_{۱۶}$ +$(۴۵۰۶۹)_{۱۶}$=$(۳۶۷۲۹)_{۱۶}$- $(۴۵۰۶۹)_{۱۶}$

۵- تفاوت مراحل اجرای دستور INT n با دستور Call Subroutine چیست و اصلاً چه فایده‌ای دارد؟ (۲ نمره)

دستور INT n اول ثبات وضعیت‌ها (Flags) را در پشته (Stack) قرار می‌دهد (Push)، سپس بیت‌های T و I (مربوط به بیت اجرای تک به تک و بیت فعال‌سازی وقفه) را صفر، یعنی غیرفعال، می‌کند، از آدرس بردار وقفه (۲+ ۴xn) مقدار CS را می‌خواند (واکشی می‌کند) بعد مقدار IP را در پشته می‌ریزد، بعد مقدار IP جدید را از جدول بردار وقفه (آدرس۴ xn) می‌خواند و سپس به آدرس متشکل از: (CS و IP/EIP) که محل شروع روال وقفه است، پرش می‌کند. دستورالعمل INT معادل یک PUSHF و سپس یک CALL Far است. دستور وقفه دو بایتی است (به جز INT3 که تک بایتی است) و لذا کوتاهتر از دستور Call far است و می‌تواند ۲۵۶ بردار وقفه را اجرا کند یا معادلاً به طور غیر مستقیم به آدرس‌های از قبل چیده شده در جدول وقفه پرش غیرمستقیم داشته باشد و البته ثبات وضعیت را نیز به طور خودکار در پشته ذخیره کند.

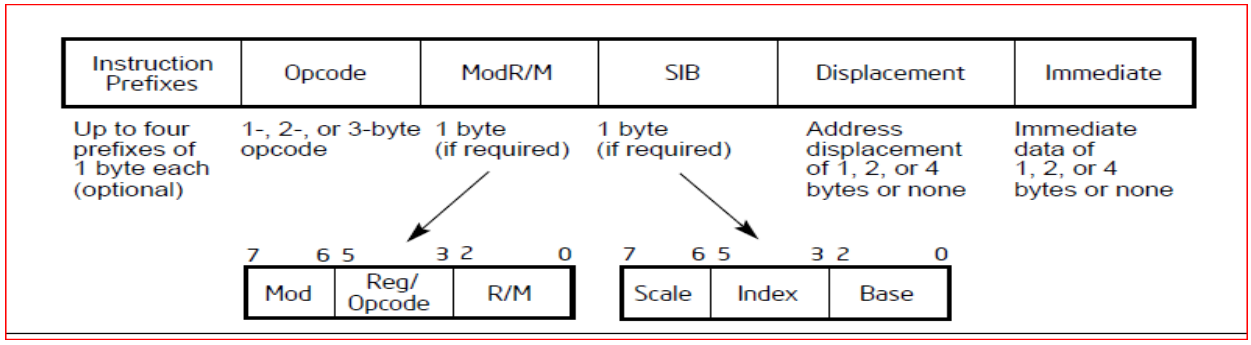
۶- این عبارت را با زبان اسمبلی فرضی یا ابداعی و معنی‌دار خود برای ۴ ماشین: بدون آدرس (پشته‌ای)، تک آدرسی (مبتنی بر انباشتگر: Accumulator)، دو آدرسی و سه آدرسی به کمک ثبات‌های دلخواه برنامه‌نویسی و سپس با هم مقایسه یا نقد کنید:

X=(A-B)/(C+D-1) (۵ نمره)

ماشین پشته‌ای	ماشین تک آدرسه	ماشین دو آدرسه	ماشین سه آدرسه
Push A	Load C	Mov Temp1,C	Mov Temp1,-1
Push B	Add D	Add Temp1,D	Add Temp1,D,Temp1
Sub	Dec	Sub Temp1,1	Add Temp1,C,Temp1
Push C	Store Temp	Mov Temp2,A	Sub Temp2,A,B
Push D	Load A	Sub Temp2,B	Div X,Temp2,Temp1
Dec	Sub B	Div Temp2,Temp1	
Add	Div Temp	Mov X,Temp2	
Div	Store X		
Pop X			

طول برنامه با تعداد ثبات‌ها یا آدرس‌های قابل استفاده در مجموعه دستورات کوتاهتر می‌شود ولی از طرف دیگر، طول دستورات نیز بزرگتر شده است چون دستورات پیچیده‌تر و حاوی آدرس‌های بیشتر شده‌اند. بنابراین نمی‌شود با قاطعیت گفت کدام برنامه تعداد بایت بیشتری دارد. البته سخت‌افزار ماشین‌های با آدرس کمتر در دستورات، ساده‌تر و معمولاً سریع‌تر است ولی طول برنامه به هر حال بیشتر است که باید به یک مصالحه تن در داد. امروزه با توجه به تعداد زیاد ثبات‌های موجود داخل پردازنده‌ها (اکثراً RISC)، حالت دو و حتی سه آدرسه بیشتر از مدل تک آدرسه (که همان ماشین مبتنی بر انباشتگر، Accumulator، است) می‌باشد. ماشین‌های پشته‌ای یا بدون آدرس نیز عمدتاً برای محاسبات ریاضی و ممیز شناور بسیار مورد استفاده است.

۷- قالب دستورات 80x86 بدین شکل است:



۷-۱ از این قالب چه مدهای آدرس‌دهی قابل استنباط است و چند تا؟ (۲ نمره)

این مدهای آدرس‌دهی قابل استنباطند:

۱-آنی (Immediate) طبق شکل، سمت راست بالا؛ ۲- ثباتی (Register) طبق میدان R/M؛ ۳- نسبی (Relative) طبق میدان Displacement شکل بالا؛ ۴- ضمنی (Implicit) طبق دستورات با Opcode تک بایتی؛ ۵- پایه (Base)؛ ۶- پایه و نمایه (Index)؛ ۷- پایه و نمایه و مقیاس‌دهی (Scale)؛ ۸- مستقیم (Direct) مطابق میدان آدرس که می‌تواند در بخش Immediate یا حتی Displacement قرار گیرد و با بیت‌های R/M تعیین تکلیف گردد؛ ۹- غیرمستقیم (مشابه استدلال قبلی).

۷-۲ طول دستورات و تعداد آن در حالت‌های مختلف چیست؟ (۲ نمره)

طولانی‌ترین دستور را می‌توان با فرض طولانی‌ترین میادین دارای (۴+۴+۱+۱+۳+۴)=۱۷ بایت دانست ولی ممکن است در عمل، برخی مُدها یا طول‌ها با هم ناسازگار باشند و تعداد بایت‌های دستورالعمل کوتاه‌تر از این مقدار باشد.
تعداد دستورات متفاوت می‌تواند به طور نظری برابر $(۲^۸)^۳$ (به خاطر ۳ بایت Opcode)، یعنی $(۲۵۶)^۳=۲^{۲۴}$ باشد که اگر با همه مدهای آدرس‌دهی ترکیب شود بالغ بر ۱۵۰ میلیارد دستور خواهد شد. البته همان‌طور که اشاره شد، برخی حالت‌ها با هم ناسازگارند و در ثانی، نیاز به این همه دستور وجود ندارد و روند طراحی پردازنده‌ها به سوی مجموعه دستورات کم (RISC)، استفاده از ثبات‌های بزرگ و پرتعداد (۳۲، یا ۶۴ عدد) و بزرگ (۳۲، ۶۴، ۱۲۸ و حتی ۲۵۶ بیتی) و معماری موازی (چند هسته‌ای) و پر سرعت است.

۷-۳ پیشوندهای دستور (Instruction Prefixes) چه کاربردی می‌تواند داشته باشد؟ (۱ نمره)

این پیشوندها اجازه می‌دهند دستورات اجرای خاص، راحت‌تر و گاه سریع‌تری توسط سخت‌افزار-نرم‌افزار داشته باشند. به عنوان مثال، rep اجازه اجرای حلقه را به طور خودکار (بر اساس مقدار CX) می‌دهد. پیشوند lock

سیگنال مربوط را در هنگام مراجعه به حافظه مشترک در گذرگاه خروجی پردازنده فعال می‌کند تا دسترسی به حافظه به صورت انحصاری انجام شود...

۸- فرض کنید برنامه‌ای نوشته‌اید که به آدرس‌های مطلق حافظه بین 1000 hex تا 2000 Hex مراجعه می‌کند و کد اجرایی (exe) آن نیز توسط اسمبلر و کامپایلر ایجاد شده است. حال اگر سیستم عامل اجازه دسترسی به این بخش از حافظه را ندهد و آدرس‌های 3FA0 به بعد را در اختیارتان گذارد "بارگذارنده" برنامه در حافظه (Loader) چکار باید بکند؟ (۲ نمره)

Loader وظیفه بارگذاری برنامه را در حافظه دارد تا از آدرس شروع آن، پردازنده مبادرت به اجرای برنامه کند. حال اگر برنامه به صورت relocatable یا غیر وابسته به آدرس‌های ثابت، نوشته نشده باشد، باید تمام آدرس‌های ثابت به آدرس‌های جدید در اختیار، تبدیل شود تا برنامه بتواند درست اجرا شود و به آدرس‌های صحیح دستور و داده‌ها دسترسی پیدا کند. در مورد صورت مسئله، آدرس3FA0 با 1000hex فاصله 2FA0 را دارد و در نتیجه همه آدرس‌های ثابت (چه مستقیم و چه غیر مستقیم) دستورات باید به همین مقدار اضافه شوند و اگر داده‌ها نیز در این فضا تعریف شده‌اند، آدرس آن‌ها نیز به همین میزان جابجا گردد.

۹- این برنامه چکار می‌کند و چطور؟ (۲ نمره)

```
        MOV BL, 08H
        MOV CX, E000H
        MOV EX, B001H
Loop:   MOV DL, [CX]
        MOV [EX], DL
        DEC BL
        JNZ loop
        HLT
```

این برنامه یک شمارنده BL را با عدد ۸ مقداردهی اولیه می‌کند و سپس در Segment داده (مورد اشاره توسط DS) داده‌ی موجود در حافظه مورد اشاره در آدرس E000h را به آدرس B001h منتقل می‌کند و این کار، با توجه به اینکه EX و CX تغییر نمی‌کند، ۸ بار عیناً تکرار می‌شود. تکرار یک عمل می‌تواند برای ایجاد یک تاخیر زمانی به صورت نرم‌افزاری معادل دستور Wait(n seconds) باشد.

۱۰- زیربرنامه زیر چکار می‌کند؟ (۲ نمره)

```
void main(void)
{
        _asm
{
        mov ah,8 ;read key no echo
        int 21h
        cmp al,‘0’ ;
        jb big
        cmp al,‘9’
        ja big
        mov dl,al ; echo (char key)
        mov ah,2
        int 21h

    big:
```

```
}
}
```

این برنامه اسمبلی مه به صورت درون خط (inline) داخل برنامه C اجرا می‌شود، تابع ۸ را به وقفه 21h رد می‌کند که به واقع خواندن از صفحه کلید است. بعد با نویسه (کاراکتر) صفر مقایسه می‌کند. اگر کوچکتر بود خارج می‌شود (چون به big پرش می‌کند) وگرنه با مقایسه میکند و اگر بزرگتر بود بازهم خارج می‌شود. در غیر این صورت (یعنی رقم دریافتی بین ۰ تا ۹ بود، آن را با صدا زدن تابع ۲ با وقفه 21h به نمایش در می‌آورد و خارج می‌شود. در واقع، نوعی کنترل ورودی قبل از نمایش آن است.

۱۱-یک برنامه ساده بنویسید که در آن برنامه اصلی (به اسمبلی) یک میانوند (Argument) را به صورت آدرس (reference, pointer) و دیگر میانوند را به صورت مقدار (Value) به زیر برنامه شما از طریق پشته منتقل کند و نتیجه را در AX پس بگیرد. (۲ نمره)

کافی است قبل از صدا زدن زیربرنامه، میانوندها را در پشته Push و سپس BP را با SP مقداردهی کنیم و بعد دسترسی‌های خود را در زیربرنامه به میانوندهای ذخیره‌شده در پشته به کمک BP انجام دهیم و قبل از مراجعه به برنامه اصلی، نتایج را یا در ثباتی که موردانتظار برنامه فراخواننده است (در اینجا، AX) یا در پشته قرار دهیم و باز گردیم.

۱۲- کاربرد Bytecode و JVM را توضیح دهید. (۲ نمره)

برای اینکه برنامه‌های جاوا بتوانند در محیط اینترنت روی همه کامپیوترها با پردازنده‌ها و مجموعه دستورات مختلف اجرا شوند نیاز به یک لایه میانی بین برنامه و پردازنده، مشابه یک ماشین مجازی، است که نقش مترجم را برای کد جاوا ایفا کند. بنابراین دستورات جاوا یه بایت‌کدهای مشخص و یکسان برای همگان ترجمه و توزیع می‌شود و این بایت‌کدها به ازای ماشین‌های مختلف به دستورات اسمبلی همان ماشین ترجمه یا تفسیر می‌گردد. پس بایت‌کدها یکسان است ولی خروجی مترجم بایت‌کد به ازای هر ماشین متفاوت. پس گ.یی یک ماشین مجازی داریم که مجموعه دستوراتش بایت‌کدهاست و برنامه جاوا به آن ترجمه می‌شود. ر هر ماشین هم یک مفسر (Interpreter) یا کامپایلر وظیفه ترجمه یا تفسیر این باین‌کدها را به کد قابل فهم و اجرا برای پردازنده مقصد به عهده دارد. بدین ترتیب برنامه یک بار برای همیشه نوشته و تبدیل به بایت‌کد استاندارد می‌شود و روی همه پردازنده‌ها اجرا می‌گردد چون هر پردازنده این بایت‌کدهای را به زبان اسمبلی و کد باینری خود تبدیل می‌کند.