# پاسخنامه تمرین سری هشتم درس معماری کامپیوتر

پاییز ۱۴۰۳

## توضیحات

- در صورت وجود ابهام یا سوال از پاسخ تمارین به تدریس یاران درس پیام دهید.

سوال ۱)

الف)

گام اول: واکشی دستورالعمل(Instruction Fetch): دستورالعملی که به PC اشاره می کند را می خواند و به پردازنده می آورد.

گام دوم: بازگشایی دستورالعمل(Instruction Decode): قسمت Opcode خوانده می شود تا نوع دستورالعمل مشخص شود.

گام سوم: خواندن عملوندها(Operands Read): با توجه به نحوه آدرس‌دهی، آدرس موثر را بدست آورده و از آن محتوای Operand ها خوانده می شود.

گام چهارم: اجرای دستورالعمل(Instruction Execute): در این مرحله از ALU استفاده می شود تا دستورالعمل اجرا شود.

گام پنجم: ذخیره کردن نتایج(Result Writeback)

گام ششم: PC = PC + ۱ و برو به گام اول

ب)

ارتباط بین پردازنده و حافظه گلوگاه این الگوی کاری است. پردازنده سرعت کاری بسیار بالاتری نسبت به حافظه دارد و هروقت که نیاز است حافظه خوانده شود، پردازنده معطل می ماند.

پ)

استفاده از حافظه نهان دستورالعمل و حافظه نهان داده

استفاده از پایپ لاین

استفاده از واحدهای پردازشی موازی برای اجرای دستورالعمل‌ها

سوال ۲)

**Clock Rates:**

- **RISC:**

معماری RISC به دلیل ساده‌تر بودن طراحی سخت‌افزار و دستورالعمل‌ها، امکان استفاده از سرعت کلاک بالاتر را فراهم می‌کند. پردازنده‌های RISC معمولاً تعداد زیادی دستورالعمل ساده را در هر سیکل اجرا می‌کنند.

- **CISC:**

معماری CISC به دلیل پیچیدگی بیشتر دستورالعمل‌ها و نیاز به زمان بیشتر برای اجرای برخی از آن‌ها، عموماً سرعت کلاک پایین‌تری دارد. هر دستور ممکن است در چندین سیکل اجرا شود.

**Design Complexity:**

- **RISC:**

طراحی پردازنده‌های RISC ساده‌تر است زیرا دستورالعمل‌ها کوچک و یکنواخت هستند. این سادگی باعث کاهش زمان و هزینه توسعه می‌شود.

- **CISC:**

پردازنده‌های CISC طراحی پیچیده‌تری دارند زیرا شامل مجموعه‌ای از دستورالعمل‌های پیچیده و متنوع هستند که برخی از آن‌ها کارهای زیادی را در یک دستور انجام می‌دهند.

**Hardware Complexity:**

- **RISC:**

در معماری RISC، سخت‌افزار ساده‌تر است زیرا بیشتر عملیات توسط نرم‌افزار انجام می‌شود. تعداد ترانزیستورها کمتر بوده و تمرکز بر سرعت و کارایی است.

- **CISC:**

پردازنده‌های CISC دارای سخت‌افزار پیچیده‌تری هستند زیرا مسئولیت اجرای دستورالعمل‌های پیچیده مستقیماً بر عهده سخت‌افزار است. این پیچیدگی می‌تواند مصرف انرژی را افزایش دهد.

**Instruction Complexity:**

- **RISC:**

دستورالعمل‌های RISC ساده، ثابت و کوتاه هستند. هر دستورالعمل معمولاً در یک سیکل کلاک اجرا می‌شود. این طراحی منجر به سرعت اجرای بیشتر می‌شود.

- **CISC:**

در معماریCISC ، دستورالعمل‌ها پیچیده‌تر و متغیر هستند. یک دستور ممکن است عملیات پیچیده‌ای را اجرا کند که گاهی نیاز به چندین سیکل کلاک دارد. این معماری برای کاهش اندازه کد طراحی شده است.

سوال ۳)

الف) آدرس‌دهی مستقیم

ب) آدرس‌دهی ثباتی غیر مستقیم

ج) آدرس‌دهی شاخص دار

د) آدرس‌دهی ثبات پایه

سوال ۴)

۱- ثبات‌های کنترلی و داده‌های لازم در طراحی این پردازنده:

با توجه به Instruction set در مورد حافظه و ثبات‌های کنترلی تصمیم گیری می‌کنیم. از اطلاعات سوال داریم که حافظه ما 1KB x 8 است. توضیحات اندازه هر کدام از ثبات هر در بخش زیر آمده است:

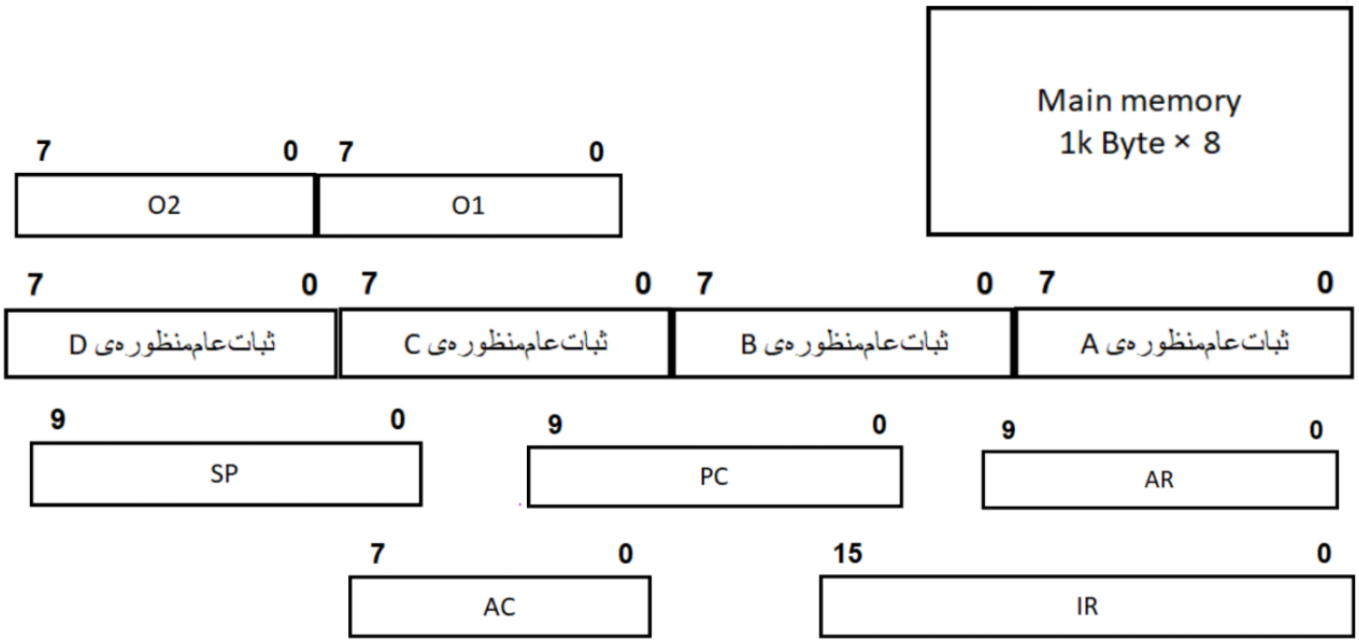
۴ تا ثبات‌های عام‌منظوره ۸ بیتی هستند و در صورت مسئله گفته شده که ۸ بیتی و برای داده هستند.

ثبات AR وظیفه نگهداری آدرس‌ها در حافظه را بر عهده دارد. با توجه به اینکه حافظه سیستم دارای ظرفیت یک کیلوبایت (معادلل ۲ به توان ۱۰ ردیف) است، این ثبات باید بتواند آدرس‌های ۱۰ بیتی را نگهداری کند. ثبات PC و ثبات SP کار نگهداری آدرس دستورالعمل‌های برنامه را ایفا می‌کند و طول آن نیز ۱۰ بیت است.

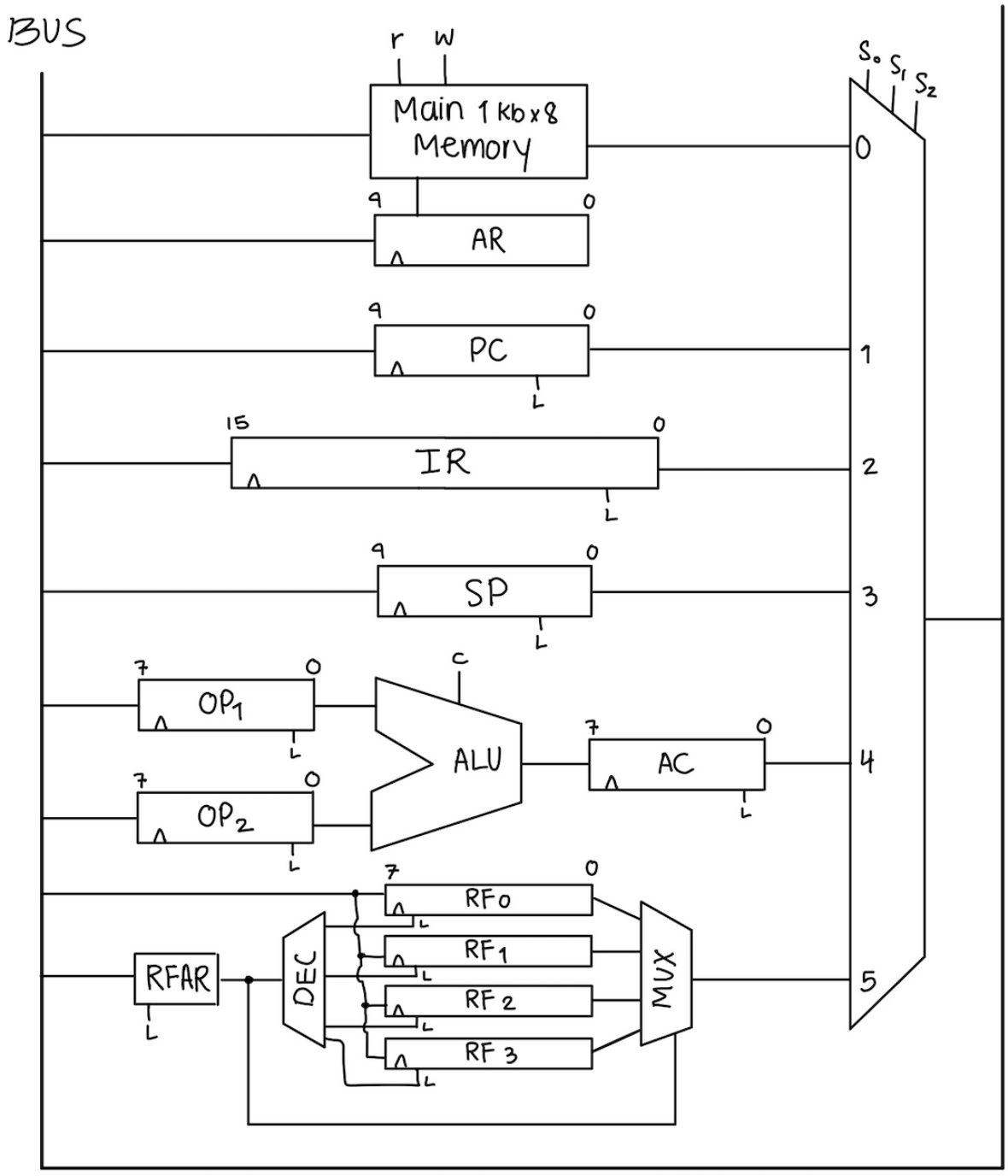
برای اجرای دستورالعمل‌ها، سیستم نیاز به ذخیره اطلاعات دستورات دارد. هر دستورالعمل ۱۶ بیت است که می‌توان آن را به کمک دو ثبات ۸ بیتی ($IR_1$ و $IR_2$) یا یک ثبات ۱۶ بیتی ذخیره کرد. اندازه این ثبات باید ضریبی صحیح از طول کلمه دستورالعمل باشد تا اطلاعات را به درستی مدیریت کند.

ثبات AC برای نگهداری خروجی ALU به کار می‌رود. از آنجایی که این ثبات تنها یک مقدار داده را در خود نگهداری می‌کند، طول آن برابر با ۸ بیت است.

ثبات‌های O1 و O2 برای نگهداری ورودی‌های ALU استفاده می‌شوند. از آنجایی که ALU برای انجام عملیات نیاز به دو ورودی دارد و هر ورودی ۸ بیت است، این ثبات‌ها نیز ۸ بیتی طراحی می‌شوند.



۲- طراحی مسیر داده پردازنده:



۳- ریز عملیات مربوط به هر دستور:

Instruction fetch:

T0: AR ← PC

T1: IR[0:8] ← M[AR], PC += 1

T2: AR ← PC

T3: IR[8:16] ← M[AR], PC += 1

Instruction decode:

T4: decode IR[14:16]

ADD:

T5.D0: AR ← IR[4:14]

T6.D0: O1 ← M[AR]

T7.D0: RFAR ← IR[0:4]

T8.D0: O2 ← RF[RFAR]

T9.D0: AC ← O1 + O2

T10.D0: M[AR] ← AC, sc ← 0

STR:

T5.D1: AR ← IR[4:14]

T6.D1: M[AR] ← IR[0:4], sc ← 0

PUSH immediate:

T5.D2: AR ← SP

T6.D2: M[AR] ← 0000:IR[0:4], SP -= 1, sc ← 0

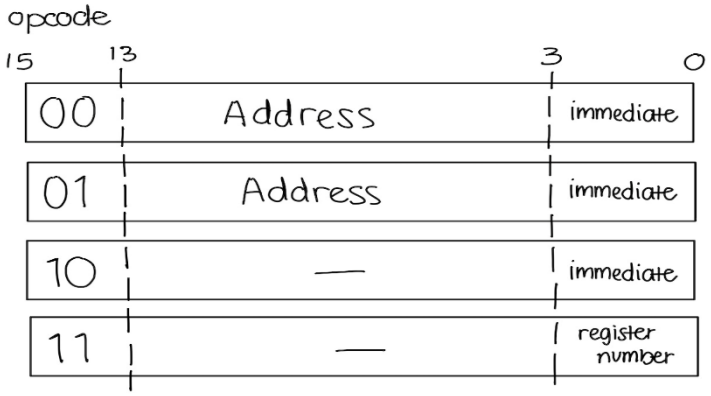
PUSH register:

T5.D3: RFAR ← IR[0:4]

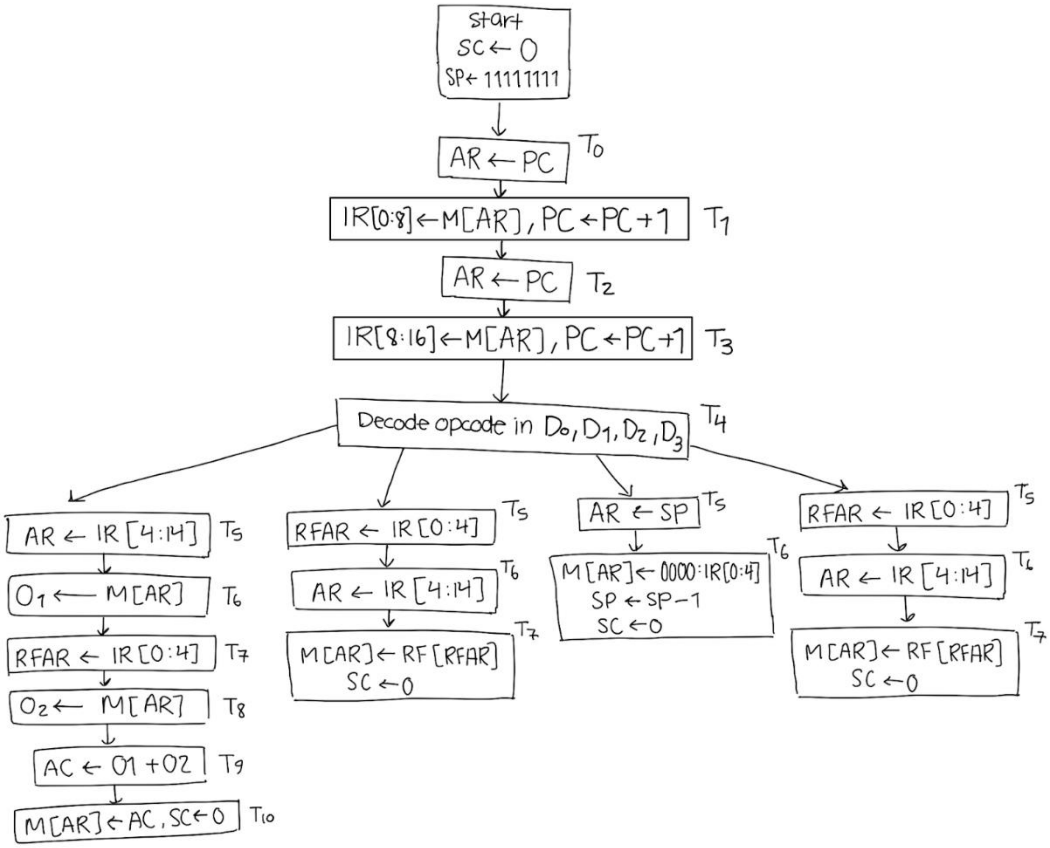
T6.D3: AR ← SP

T7.D3: M[AR] ← RF[RFAR], SP -= 1, sc ← 0

۴- طراحی قالب دستورالعمل:




۵- فلوچارت:



۶- طراحی واحد کنترل:

increment_PC=T1+T3

decrement_SP=T6D2+T7D3

clear_sc=T7D3+T6D2+T6D1+T7D0

PC_Load: 0

IR_Load: T1 + T3

O1_Load: T6.D0

O2_Load: T8.D0

AC_Load: T9.D0

SP_Load: T4

M_Read: T1 + T3 + T6.D0

M_Write: T10.D0 + T6.D1 + T6.D2 + T7.D3

COMMAND: T9.D0

۶ تا ورودی مختلف به گذرگاه کلی داریم، پس به ۳ بیت برای انتخاب هر کدام از این حالت‌ها نیاز داریم. برای این به یک دیکدر نیاز داریم که این ۳ بیت را دریافت کرده و خروجی لازم را در مواقع درست تولید کند. ورودی‌های آن را با $x_0$ تا $x_7$ نشان می‌دهیم.

| سیگنال | مقدار بیت‌ها ($x_0$ تا $x_7$) | $S_2$ $S_1$ $S_0$ |
|---|---|---|
| Memory | ۰۰۰۰۰۰۰۱ | ۰۰۰ |
| PC | ۰۰۰۰۰۰۱۰ | ۰۰۱ |
| IR | ۰۰۰۰۰۱۰۰ | ۰۱۰ |
| AC | ۰۰۰۰۱۰۰۰ | ۰۱۱ |
| SP | ۰۰۰۱۰۰۰۰ | ۱۰۰ |
| MUX | ۰۰۱۰۰۰۰۰ | ۱۰۱ |

$x_0 = T1 + T3 + T6.D0$

$x_1 = T0 + T2$

$x_2 = T5.D0 + T7.D0 + T5.D1 + T6.D1 + T6.D2 + T5.D3$

$x_3 = T10.D0$

$x_4 = T5.D2 + T6.D3$

$x_5 = T8.D0 + T7.D3$

$x_6 = 0$

$x_7 = 0$

7- برنامه اسمبلی برای جمع کردن اعداد ۱ تا ۳:

```
01 0001000000 0000 str [64], 0

00 0001000000 0001 add [64], 1

00 0001000000 0010 add [64], 2

00 0001000000 0011 add [64], 3
```