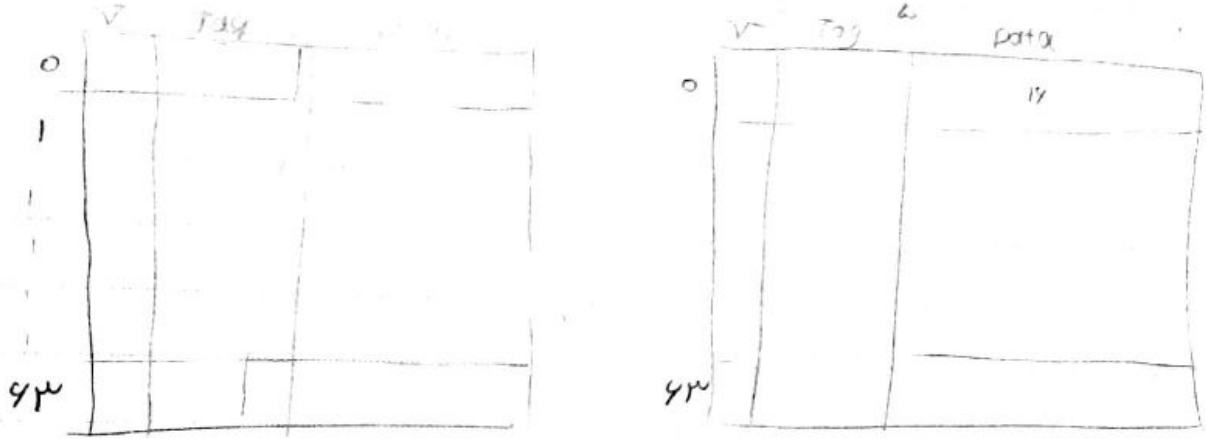
Subject cache
Date

2-way associative



1 bit valid + ۴ bit Tag + ۱۶ bit data = ۲۱

اندازه حافظه : ۲(۲۱) = ۴۲

۲(۱۶) = ۴۲ (۸)

تعداد ردیف‌های index = ۲
cache
۲^۶ = ۶۴
ردیف

Set-0 : 64×21
Set-1 : 64×21

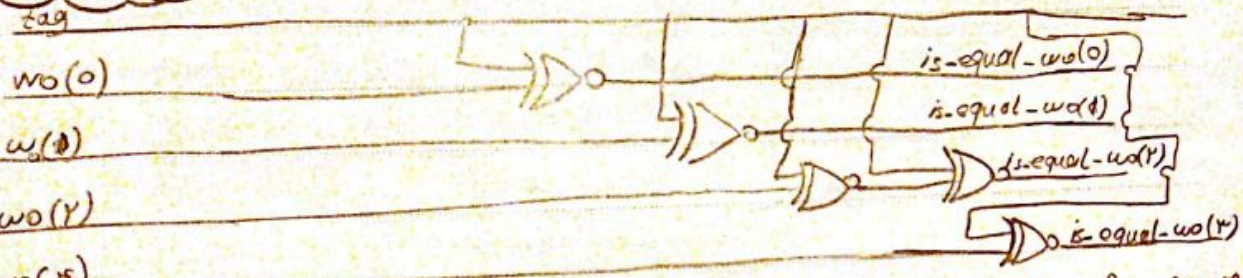
دو مجموعه داریم که هرکدام حاوی ۱۶ بیت دیتا هستند

d

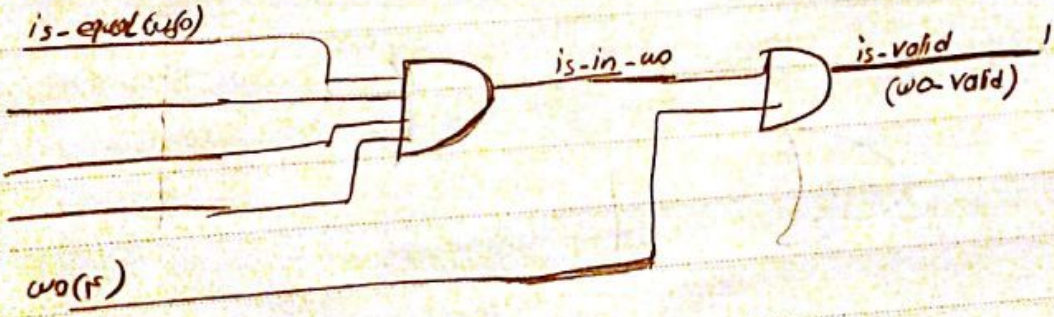
Subject

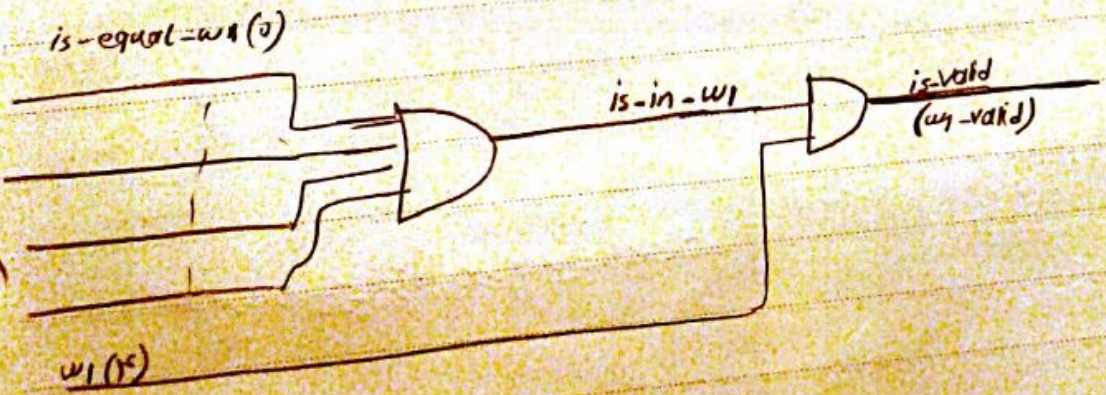
Date

Miss-Hit-Logic



A	B	A xnor B
0	0	1
0	1	0
1	0	0
1	1	1





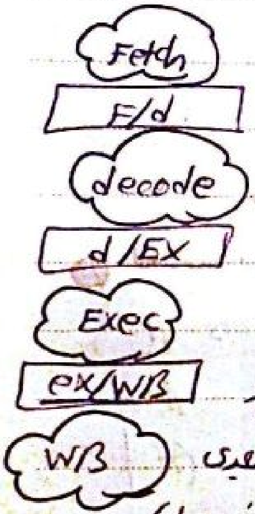
Subject: a Hazard control unit for SAYEH ۹۴/۳/۲۲

Date:

بخش دوم پروژه
طراحی پایپ لاین برای سایه

1. Draw sayeH pipeline
2. How many registers do you add for creating pipeline?
3. which type of Hazards do we have in SAYEH Pipeline?
4. Design a hazard controller unit.

طراحی پایپ لاین شامل این اقدامات است که رجیسترهایی بین هر دو مرحله ی پشت سر هم قرار می دهیم.



سه رجیستر اضافه می شود

(اگر بخش دسترسی به حافظه را در نظر بگیریم، می توان برای راحتی کار، چهار رجیستر قرار داد)

بین هر دو مرحله، یک رجیستر قرار داده می شود که نام گذاری آن ها بر اساس دو مرحله قبل و بعد است. داده هایی که در هر مرحله تولید می شود، در این رجیستر قرار داده می شود تا به مراحل بعدی انتقال داده شود. (بعد از WB دیگر نتایج در رجیسترفایل نوشته شده و نیازی به رجیستر نیست!)

اصلی ترین راه حل رفع هازارد: stall (تأخیر در اجرای دستورات)

و در ادامه: پر کردن این stall ها در هنگام execution stage، با Nop.

↓

Bubble هم

حفظیات

توجه: در معماری کامپیوتر میپس، طراحی اولیه به صورت سینگل سایکل بود، که در آن، حافظه ی دستور و داده، جدا بودند. در معماری مولتی سایکل، حافظه دستور و داده، یکی هستند. از آنجا که در کلاس، معماری سینگل سایکل برای کامپیوتر میپس در بخش طراحی پایپ لاین مورد بررسی قرار گرفت، لذا در طراحی پایپ لاین برای سایه هم ما طبق همین روند کلاس جلو رفتیم، و حافظه ی دستور و داده را جداگانه در نظر گرفتیم.

Subject

Date

راه حل ←

1) stall
2) delayed Load (جابجایی دستورات)
3) Nop
4) operand Fw
5) تاخیر یا دو مرحله

انواع مخاطرات خط لوله (هازاردها):

1) Data Hazard ⇐ دلیل: Data Dependency
   1) RAW
   2) WAR
   3) WAW

2) structural Hazard ⇐ دلیل: resource conflict ⇐ راه حل: دو حافظه (جدا/مستقل) (cache و ...)
   - Ram ⇒ Data
   - cache ⇒ Instruction

3) control Hazard ⇐ دلیل: Branch Difficulties

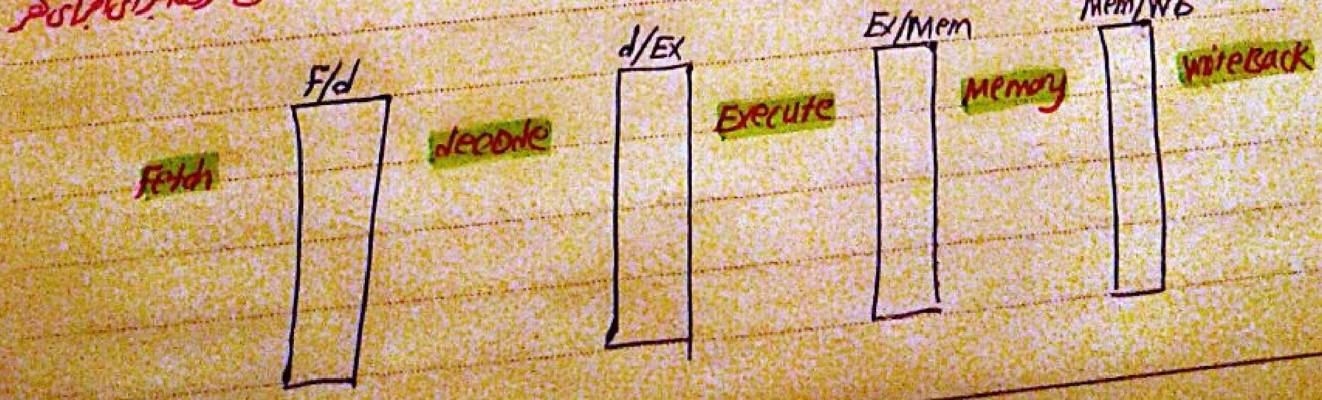
راه حل ←

1) stall
2) flush
3) Branch target prediction (idle)
4) Delayed Branch ← جابجایی دستورات
5) Branch target Buffer (← شامل Fetch، یک حافظه اضافه کنیم که تمام دستورهای انشعاب در آن ذخیره شوند)
6) Noop

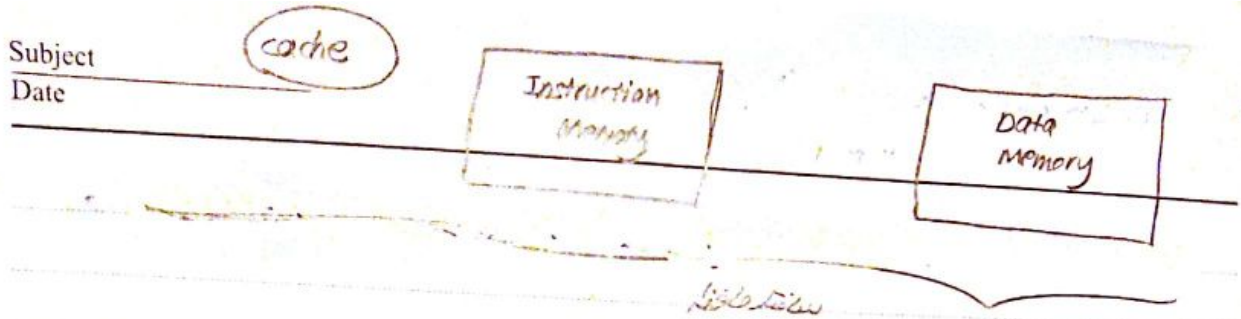
V.3

1) Draw pipeline
2) number of Registers
3) Type of hazards
4) hazard controller unit

در نهایت اگر طبق روند کد درس بخواهیم جلو برویم، پایپ لاین ما نیاز به ۴ رجیستر دارد و پنج مرحله برای اجرای هر دستور خواهیم داشت.



Subject
Date



Fetch

pc + memory + addressing + instruction memory

pc + add unit

deede

IR + WP + Register File

Execute

ALU + address Logic + FWD

Memory

Data Memory

write Back

(۴)

قسمت سوم:

بررسی سه نوع وابستگی داده‌ها:

* حین خواندن دستور و داده یک است، فقط هزارد structural داریم (چون دسترسی همزمان به حافظه صحبت پیش می‌آید).

با توجه به اینکه cache داریم، این مشکل قابل حل است.

می‌توان گفت داده‌ها را از هم بخواند و دستورات را از کش.

اگر اطلاعات نیز نبود، می‌توان حافظه‌ی سریع دیگری قرار داد.

در مورد بررسی کوهازارد دیگر، باید به دستورات نگاه کنیم.

nop	
hlt	mil
szf	mih
czf	sap (save pc)
scf	
ccf	jpa
cwp	jpr
mvr	brz (branch if zero)
lda	brc (branch if carry)
sta	
inp	awp (wp=wp+1)
oup	
and	
orr	
not	
shl	
shr	
add	
sub	
mul	
comp	

برای دستورات jump و branch، احتمال بروز control Hazard وجود دارد.
(branch difficulties)

راه حل: راه‌حل‌هایی مانند noop، یا پیش‌بینی Branch، کار کامپایلر است. (برای قبل از این دستورات stall بگذاریم)

اگر بخواهیم یک راه‌حل ساده پیشنهاد بدهیم، حل استفاده از Buffer سرراست‌ترین کار است.
Branch target Buffer.

یک حافظه‌ی سریع در کنار بخش fetch اضافه می‌کنیم تا دستورات Branch را در آن ذخیره کنیم.

البته از روش‌هایی مانند stall دادن (و گذاشتن دستور Noop) که هم می‌توان استفاده کرد
حتی از روش جابجایی دستورات و یا idle (Branch prediction)

بخش Branch target Buffer را در کنار بخش fetch قرار می‌دهیم و برای دستورات Branch باید به این قسمت رجوع کنیم)

سیگنال‌های کنترلی کنترلر نیز باید متناسب با همین فاز کنید مقداردهی شوند.

بررسی Data Hazard :

mvr	RD ⇐ RS
lda	RD <= (RS)
sta	(RD) ⇐ RS
inp	⇒ from RS to RD
oup	RD ⇒ from RS to RD
and	RD ⇐ RD & RS
orr	RD ⇐ RD / RS
not	RD <= ~ RS
shl	RD <= sla RS
shr	RD ⇐ sra RS
add	RD ⇐ RD + RS + C
sub	RD ⇐ RD − RS − C
mul	RD ⇐ RD * RS (8bit)
cmp	RD , RS (مقایسه)

mvl : RDL ⇐ {8bit , I}
mih RDH <= {I , 8bit Z}

در این جا به بیشتر هم قرار گرفتن بعضی از دستورات، ممکن است مشکل ایجاد شود مثلا:

$$\begin{cases} add: & RD \Leftarrow RD + RS + C \\ sub: & RD \Leftarrow RD - RS - C \end{cases} \Rightarrow WAW$$

و یا

$$\begin{cases} RD \Leftarrow RD / RS & (And) \\ RD \Leftarrow \sim RS & (not) \end{cases} \Rightarrow WAW$$

راه حل:

استفاده از دستور NOP (که جزو دستورات خود سیمه لعر هست!)

یا روشن کارتر آن است که از operand forwarding استفاده شود (هنگام پیش‌بینی شود)

مداری طراحی می‌شود که ورودی و خروجی را تشخیص می‌دهد

پس یک operand fw unit در کنار بخش fetch قرار می‌گیرد.

Data Hazards

RAW ✓ WAW ✓ WAR ✗

Subject
Date

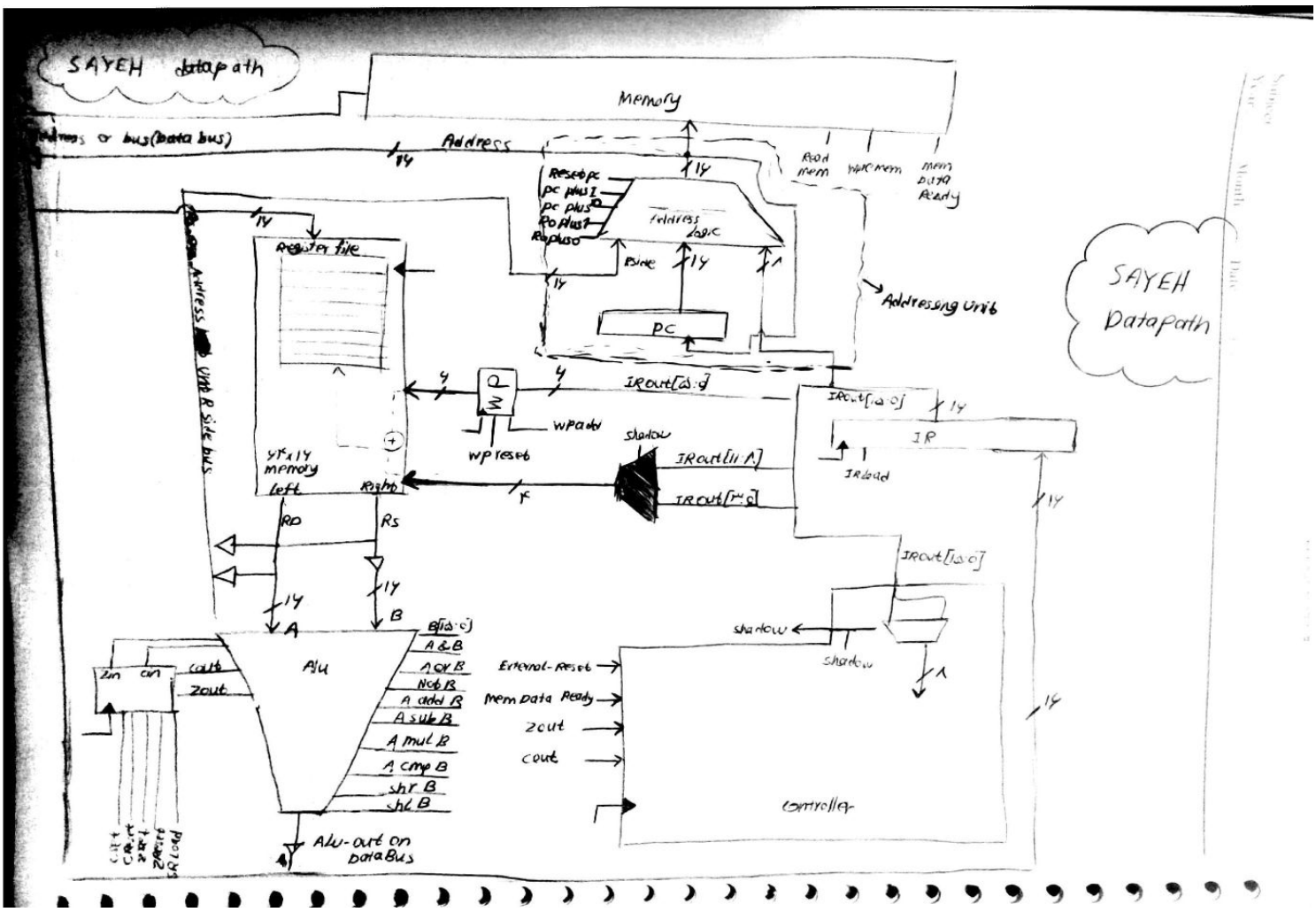
بخش‌هایی که باید اضافه بشوند:

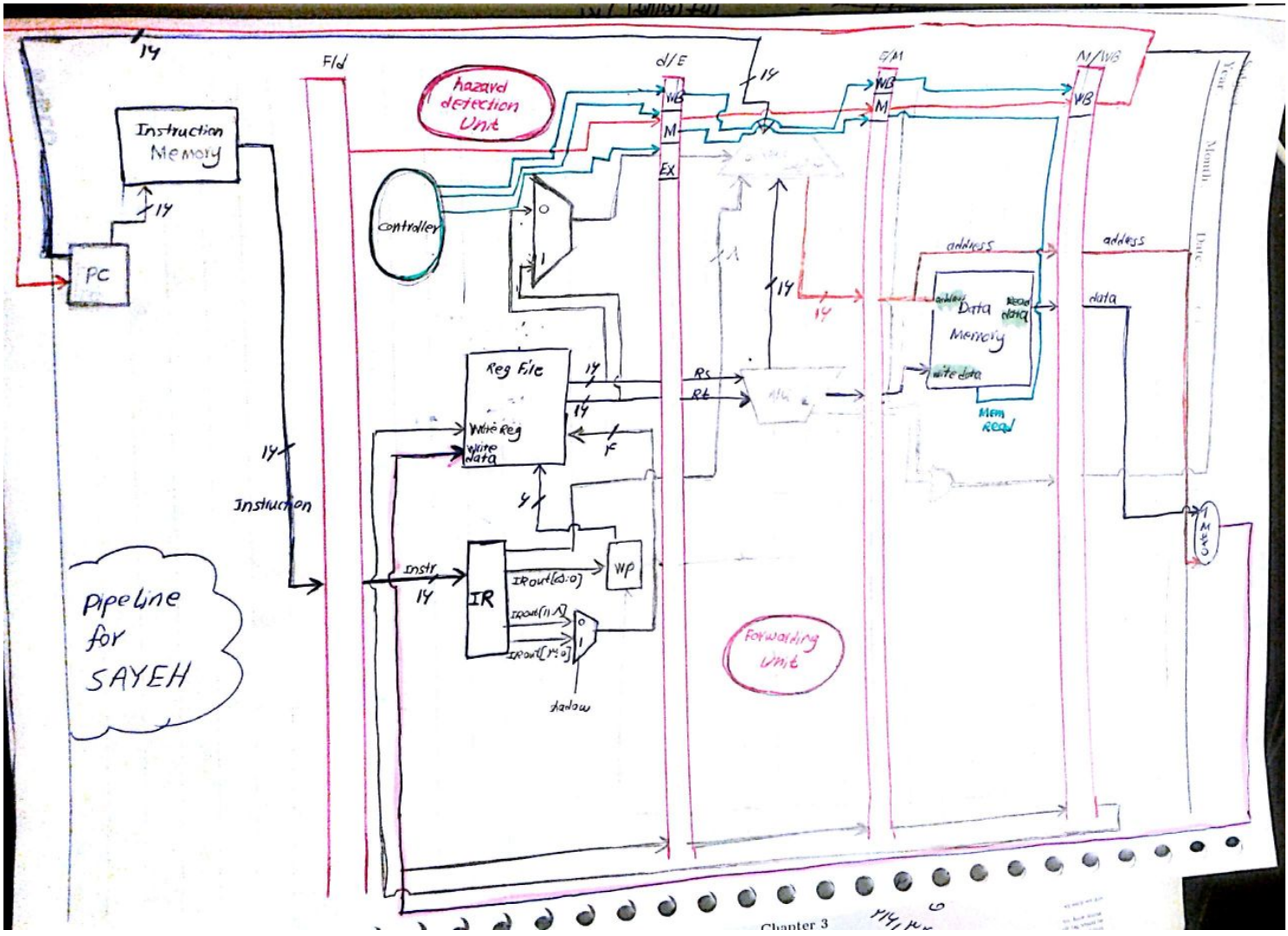
1. Hazard detection unit ⇒ در کل برای تشخیص هزارد
2. forwarding unit ⇒ در بخش fetch ⇒ برای Data Hazard

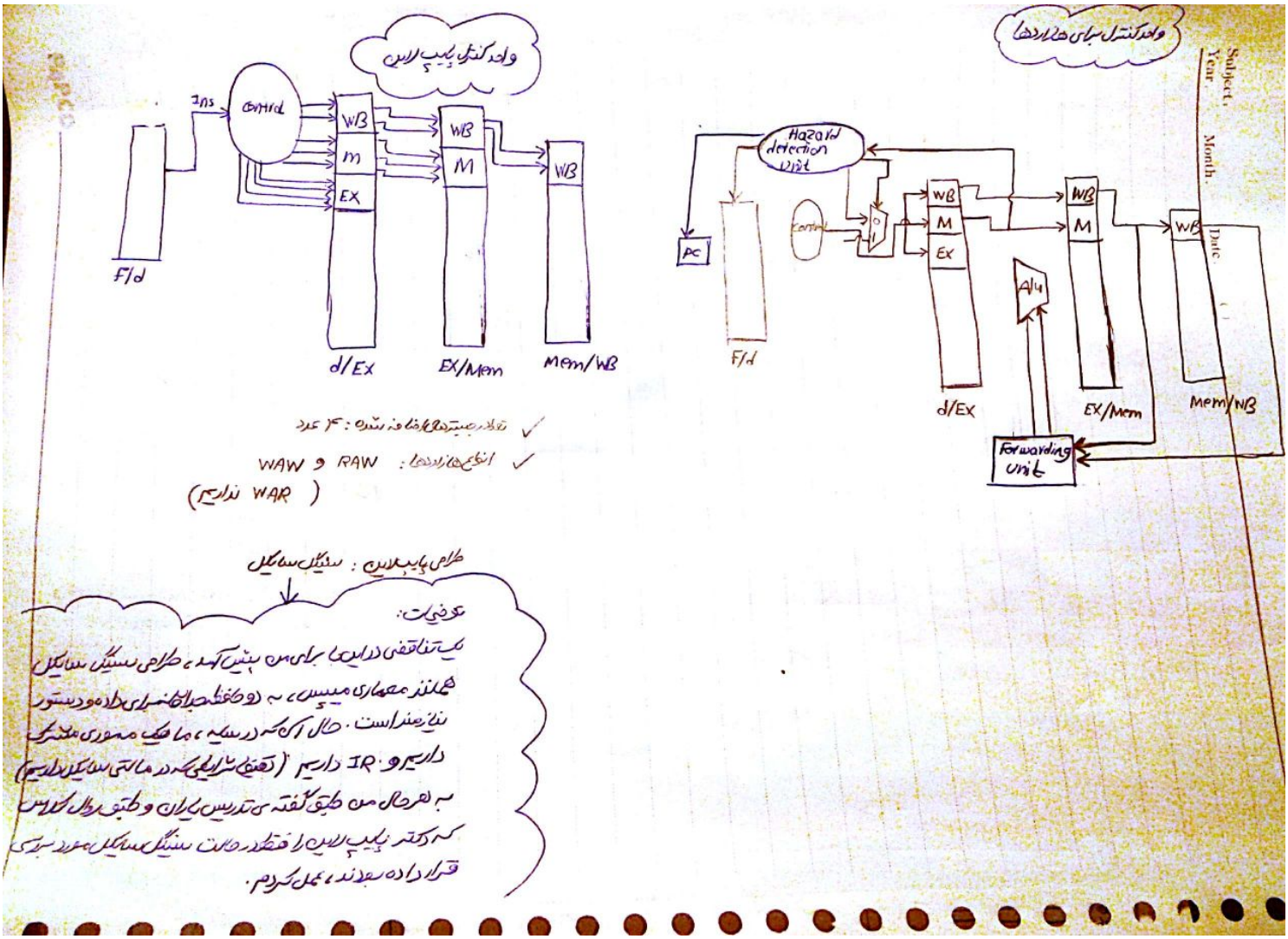
3. یک حافظه‌ی سریع (Branch target Buffer) در بخش fetch
4. در صورت لزوم یک حافظه سریع برای رفع مشکل structural هزارد (که بعید است اگر هزاردها را درست مدیریت کرده باشیم، ایجاد شود)

تذکرات:

- ✓ ۳۲ رجیستر اضافه شده برای پایپ‌لاین
- ✓ بررسی انواع هزاردهای به وجود آمده
- ✓ رسم پایپ‌لاین
- ✓ طراحی واحد کنترلر برای هزاردها (پایپ‌لاین جدیدی به همراه سیگنال کنترلی دارد)







✓ تعداد دستورالعمل‌های فرضی شده: ۴ عدد

✓ انواع هازاردها: RAW و WAW

(WAR نداریم)

طراحی پایپ‌لاین: سینگل سایکل

توضیحات:

یک تناقضی (دارایی) برای من پیش آمد، طراحی سینگل سایکل همانند معماری میپس، به دو حافظه جداگانه برای داده و دستور نیاز مند است. حال آن که در سیکه، ما فقط معماری مشترک داریم و IR داریم (تفاوتی که در مباحث سینگل سایکل داریم) به هر حال من طبق گفته‌ی تدریس یاران و طبق روال کلاس که گفته پایپ لاین را فقط در حالت سینگل سایکل مورد بررسی قرار داده بودند، عمل کردم.