Review Questions

4.1 What are the differences among sequential access, direct access, and random access?

For the sequential access, it always starts at current location to the desired location in a specific linear sequence. Its access time only depends on the physical location.

For the direct access, individual blocks or records have a unique address based on physical location and it will first jump to a vicinity place then search sequentially. Its access time depends on not just physical location, but also the previous access.

For the random access, all individual addresses identify a location exactly, any location can be selected at random and directly addressed. Its access time is independent of the physical location or previous access and is constant.

4.2 What is the general relationship among access time, memory cost, and capacity?

There is a trade-off among these three characteristics of memory.

Faster access time, greater cost per bit.

Greater capacity, smaller cost per bit.

Greater capacity, longer/slower access time.

4.3 How does the principle of locality relate to the use of multiple memory levels?

Memory reference tend to cluster, it has spatial locality and temporal locality, so it is possible to organize data across a memory hierarchy such that the percentage of accesses to each successively lower level is substantially less than that of the level above, that means we can put the data cluster with higher success rate to the small amount of fast memory, which can make higher average access speed to build a way out of the trade-off dilemma.

Note: All main memories are addressable on byte.

4.5 Consider a 32-bit microprocessor that has an on-chip 16-KByte four-way set-associative cache. Assume that the cache has a line size of four 32-bit words. Draw a block diagram of this cache showing its organization and how the different address fields are used to determine a cache hit/miss. Where in the cache is the word from memory location ABCDE8F8 mapped?

block size = line size = four 32-bit words = 16 bytes —> 4 bits word identifier

28bits block identifier

number of lines in cache = 16-KByte/16 bytes = 2^{10}

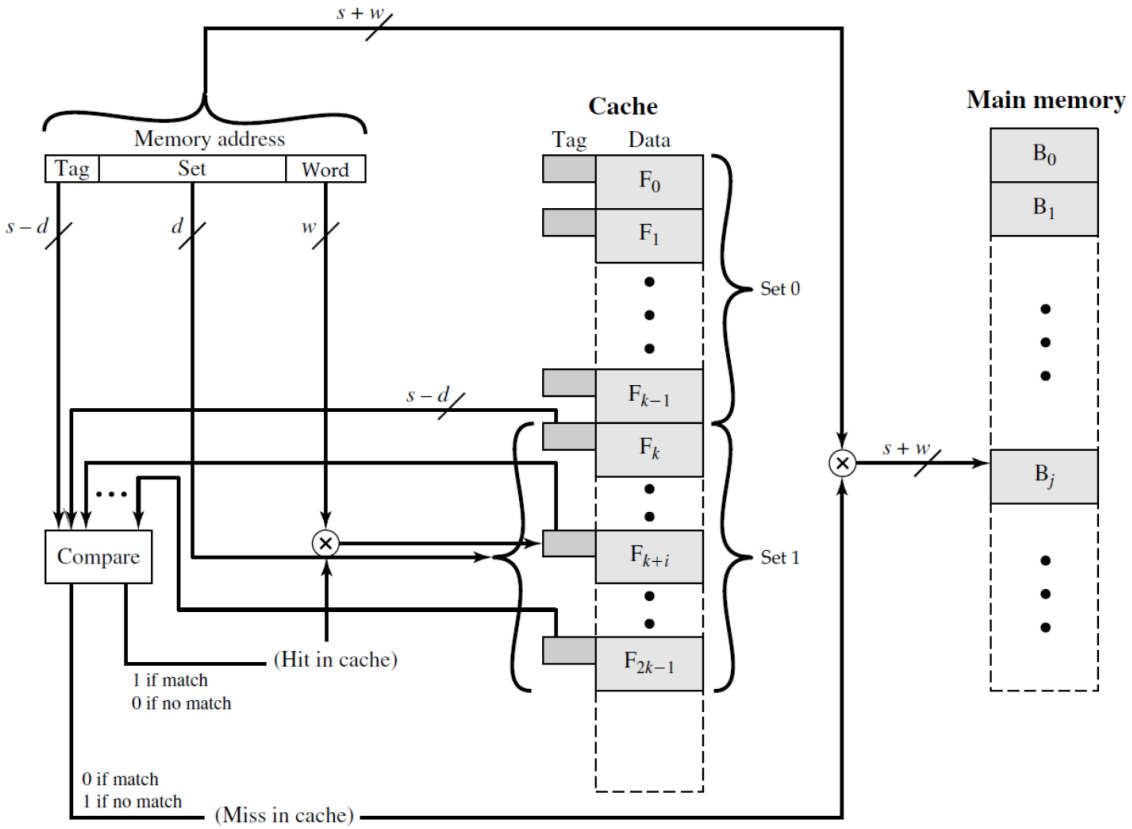
$2^{10}/4 = 2^8$ sets in cache —> 8 bits set number

28 - 8 = 20 bits tag

The address format is as follows:

Tag	Set	Word
20	8	4

The block diagram of the cache is as follows：(s=28, d=8, w=4)



A	B	C	D	E	8	F	8

The word from memory location ABCDE8F8 mapped in the cache is 8F(Set=143),any line, byte 8.

4.6 Given the following specifications for an external cache memory: four-way set associative; line size of two 16-bit words; able to accommodate a total of 4K 32-bit words from main memory; used with a 16-bit processor that issues 24-bit addresses. Design the cache structure with all pertinent information and show how it interprets the processor's addresses.

block size = line size = two 16-bit words = 4 bytes —> 2 bits word identifier

22bits block identifier

number of lines in cache = 4K 32-bit words /4 bytes = 2^{12}

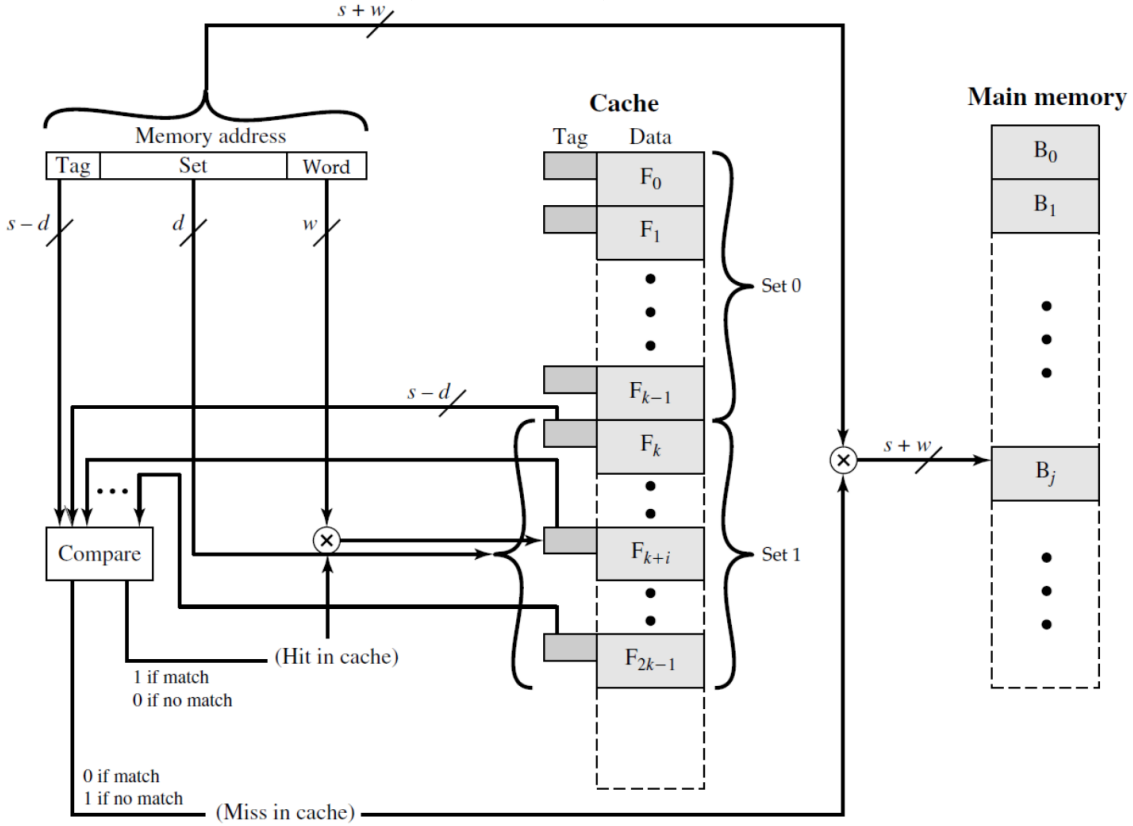
2^{12} /4 = 2^{10} sets in cache —> 10bits set number

22 - 10 = 12 bits tag

The address format is as follows:

Tag	Set	Word
12	10	2

The cache structure is as follows：(s=22, d=10, w=2)



4.7 The Intel 80486 has an on-chip, unified cache. It contains 8 KBytes and has a four-way set-associative organization and a block length of four 32-bit words. The cache is organized into 128 sets. There is a single "line valid bit" and three bits, B0, B1, and B2 (the "LRU" bits), per line. On a cache miss, the 80486 reads a 16-byte line from main memory in a bus memory read burst. Draw a simplified diagram of the cache and show how the different fields of the address are interpreted.

Number of address bits for Inter 80486 is 32.

block size = line size = four 32-bit words = 16 bytes —> 4 bits word identifier

32-4=28bits block identifier

128 sets in cache —> 7bits set number

28 - 7 = 21 bits tag

The address format is as follows:

Tag	Set	Word
21	7	4

Each set in the cache includes 3 LRU bits and four lines. Each line consists of 4 32-bit words, a valid bit, and a 21-bit tag.

(题目中说法有误, there are three bits, B0, B1, and B2 (the "LRU" bits), per set.)

4.8 Consider a machine with a byte addressable main memory of 216 bytes and block size of 8 bytes. Assume that a direct mapped cache consisting of 32 lines is used with this machine.

a. How is a 16-bit memory address divided into tag, line number, and byte number?

Tag	Line	Word
8	5	3

block size = line size = 8 bytes —> 3 bits word identifier

16-3=13 bits block identifier

32 lines in cache —> 5 bits line number

13 - 5 = 8 bits tag

A 16-bit memory address is divided into 8 bits tag, 5 bits line number and 3 bits word identifier.

b. Into what line would bytes with each of the following addresses be stored?

0001 0001 0001 1011 —> 00011 —> Line 3

1100 0011 0011 0100 —> 00110 —> Line 6

1101 0000 0001 1101 —> 00011 —> Line 3

1010 1010 1010 1010 —> 10101 —> Line 21

c. Suppose the byte with address 0001 1010 0001 1010 is stored in the cache. What are the addresses of the other bytes stored along with it?

The addresses of other bytes stored with it are as follows:

0001 1010 0001 1000; 0001 1010 0001 1001; 0001 1010 0001 1011; 0001 1010 0001 1100

0001 1010 0001 1101; 0001 1010 0001 1110; 0001 1010 0001 1111

d. How many total bytes of memory can be stored in the cache?

32 lines * 8 bytes = 256 bytes

e. Why is the tag also stored in the cache?

In a direct mapped cache, each line corresponds to multiple blocks in main memory. When a block is actually read into its assigned line, it is necessary to tag the data to distinguish it from other blocks that can fit into that line. So, the tag is also stored in the cache.

4.11 Consider a memory system that uses a 32-bit address to address at the byte level, plus a cache that uses a 64-byte line size.

a. Assume a direct mapped cache with a tag field in the address of 20 bits. Show the address format and determine the following parameters: number of addressable units, number of blocks in main memory, number of lines in cache, size of tag.

The address format is as follows:

Tag	Line	Word
20	6	6

block size = line size = 64 bytes —> 6 bits word identifier

20 bits tag

32 – 20 – 6 = 6 bits line number

number of addressable units = 2^{32} bytes;

number of blocks in main memory = 2^{26};

number of lines in cache = $2^6 = 64$;

size of tag = 20 bits.

b. Assume an associative cache. Show the address format and determine the following parameters: number of addressable units, number of blocks in main memory, number of lines in cache, size of tag.

The address format is as follows:

Tag	Word
26	6

block size = line size = 64 bytes —> 6 bits word identifier

32 – 6 = 20 bits tag

number of addressable units = 2^{32} bytes;

number of blocks in main memory = 2^{26};

number of lines in cache = undetermined;

size of tag = 26 bits.

c. Assume a four-way set-associative cache with a tag field in the address of 9 bits. Show the address format and determine the following parameters: number of addressable units, number of blocks in main memory, number of lines in set, number of sets in cache, number of lines in cache, size of tag.

The address format is as follows:

Tag	Set	Word
9	17	6

block size = line size = 64 bytes —> 6 bits word identifier

9 bits tag

32 – 9 – 6 = 17 bits set number

number of addressable units = 2^{32} bytes;

number of blocks in main memory = 2^{26};

number of lines in set = $2^2 = 4$;

number of sets in cache = 2^{17};

number of lines in cache = 2^{19};

size of tag = 9 bits.

4.22 A computer has a cache, main memory, and a disk used for virtual memory. If a referenced word is in the cache, 20 ns are required to access it. If it is in main memory but not in the cache, 60 ns are needed to load it into the cache, and then the reference is started again. If the word is not in main memory, 12 ms are required to fetch the word from disk, followed by 60 ns to copy it to the cache, and then the reference is started again. The cache hit ratio is 0.9 and the main memory hit ratio is 0.6. What is the average time in nanoseconds required to access a referenced word on this system?

location	probability	time(ns)
In Cache	0.9	20
In Main memory Not in cache	0.1*0.6	60+20
In disk Not in cache and main memory	0.1*0.4	12000000+60+20

The average access time is:

T_{avg} = 0.9*20ns + 0.1*0.6*(60+20) ns + 0.1*0.4*(12*10^6+60+20) ns = 480026 n

Problems: 12.6, 12.18, 12.19

12.6 Compare zero-, one-, two-, and three-address machines by writing programs to compute

$$X = (A + B * C)/(D - E * F)$$

0 Address	1 Address	2 Address	3 Address
PUSH M	LOAD M	MOVE (X ← Y)	MOVE (X ← Y)
POP M	STORE M	ADD (X ← X + Y)	ADD (X ← Y + Z)
ADD	ADD M	SUB (X ← X − Y)	SUB (X ← Y − Z)
SUB	SUB M	MUL (X ← X × Y)	MUL (X ← Y × Z)
MUL	MUL M	DIV (X ← X/Y)	DIV (X ← Y/Z)
DIV	DIV M		

for each of the four machines. The instructions available for use are as follows:

reverse Polish：ABC*+DEF*-/

0 ADDRESS	1 ADDRESS	2 ADDRESS		3 ADDRESS	
PUSH A	LOAD E	Instruction	comment	Instruction	comment
PUSH B	MUL F	MOVE X, B	X ←B	MUL X, B, C	X← B * C
PUSH C	STORE Y	MUL X, C	X← X*C	ADD X, A, X	X← A+X
MUL	LOAD D	ADD X, A	X←X+A	MUL Y, E, F	Y←E*F
ADD	SUB Y	MOVE Y, E	Y←E	SUB Y, D, Y	Y←D-Y
PUSH D	STORE Y	MUL Y, F	Y←Y*F	DIV X, X, Y	X←X/Y
PUSH E	LOAD B	MOVE Z, D	Z←D	答案不唯一	
PUSH F	MUL C	SUB Z, Y	Z←Z-Y		
MUL	ADD A	DIV X, Z	X←X/Z		
SUB	DIV Y	答案不唯一			
DIV	STORE X				
POP X	答案不唯一				
reverse Polish					

12.18 Convert the following formulas from reverse Polish to infix:

a. AB + C + D *

b. AB/CD/ +

c. ABCDE + * */

d. ABCDE + F/ + G - H/ * +

a. (A+B+C) *D

b. A/B+C/D

c. A/(B*C*(D+E))

d. A+B*(C+(D+E)/F-G)/H

12.19 Convert the following formulas from infix to reverse Polish:

a. A + B + C + D + E

b. (A + B) * (C + D) + E

c. (A * B) + (C * D) + E

d. (A - B) * (((C - D * E)/F)/G) * H

a. AB+C+D+E+

b. AB+CD+*E+

c. AB*CD*+E+

d. AB-CDE*-F/G/*H*

Problems: 13.3, 13.4, 13.5

13.3 An address field in an instruction contains decimal value 14. Where is the corresponding operand located for:

a. immediate addressing → the address field

b. direct addressing → memory location 14

c. indirect addressing → the memory location whose address is in memory location 14

d. register addressing → register 14

e. register indirect addressing → the memory location whose address is in register 14

13.4 Consider a 16-bit processor in which the following appears in main memory, starting at location 200:

200	Load to AC	Mode
201	500	
202	Next instruction	

The first part of the first word indicates that this instruction loads a value into an accumulator. The Mode field specifies an addressing mode and, if appropriate, indicates a source register; assume that when used, the source register is R1, which has a value of 400.There is also a base register that contains the value 100.The value of 500 in location 201 may be part of the address calculation. Assume that location 399 contains the value 999, location 400 contains the value 1000, and so on. Determine the effective address and the operand to be loaded for the following address modes:

a. Direct; b. Immediate; c. Indirect; d. PC relative; e. Displacement; f. Register; g. Register indirect; h. Autoindexing with increment, using R1

	EA	Operand	
a. Direct	500	1100	EA = 500　直接可得
b. Immediate	201	500	201存储的为操作数
c. Indirect	1100	1700	EA=（500）=1100
d. PC relative	702	1302	EA=A+（PC）=500 + 202 PC = 200 + 2
e. Displacement	600	1200	EA=A+(R) = 500+100 R为base register
f. Register	R1	400	EA = R1　直接可得
g. Register indirect	400	1000	EA= （R1）=400
h. Autoindexing with increment, using R1	900	1500	EA=A+(R1) =900 R1← R1 + 1

13.5 A PC-relative mode branch instruction is 3 bytes long. The address of the instruction, in decimal, is 256028. Determine the branch target address if the signed displacement in the instruction is –31.

relative addressing: EA = A + (PC)

PC = 256028 + 3 = 256031, A= –31

EA = 256031 – 31 = 256000

Review Problems: 14.5, 14.6, 14.7

14.5 Why is a two-stage instruction pipeline unlikely to cut the instruction cycle time in half, compared with the use of no pipeline?

(1). The execution time will generally be longer than the fetch time. Execution will involve reading and storing operands and the performance of some operation. Thus, the fetch stage may have to wait for some time before it can empty its buffer.

(2). A conditional branch instruction makes the address of the next instruction to be fetched unknown. Thus, the fetch stage must wait until receives the next instruction address from the execute stage. The execute stage may then have to wait while the next instruction is fetched.

14.6 List and briefly explain various ways in which an instruction pipeline can deal with conditional branch instructions.

(1). Multiple streams: Using two pipelines and allowing pipelines to fetch both instruction which are choices of a branch instruction.

(2). Prefetch branch target: Prefetching the target of the branch and saving it until the branch instruction is executed.

(3). Loop buffer: Using a small, very high speed memory to maintain the n most recently fetched instruction in sequence, which will be first checked when a branch is occurred.

(4). Branch prediction: Using static or dynamic approaches to predict whether a branch will be taken.

(5). Delayed branch: Automatically rearranging instructions within a program. Branch occurs later than desired.

14.7 How are history bits used for branch prediction?

One or more bits that reflect the recent history of the instruction can be associated with each conditional branch instruction. These bits are referred to as a taken/not taken switch that directs the processor to make a particular decision the next time the instruction is encountered.

Questions: 14.7, 14.8, 14.11

14.7 Consider the timing diagram of Figures 14.10. Assume that there is only a two-stage pipeline (fetch, execute). Redraw the diagram to show how many time units are now needed for four instructions.

	1	2	3	4	5
Instruction 1	FI	EI			
Instruction 2		FI	EI		
Instruction 3			FI	EI	
Instruction 4				FI	EI

14.8 Assume a pipeline with four stages: fetch instruction (Fl), decode instruction and calculate addresses (DA), fetch operand (FO), and execute (EX). Draw a diagram similar to Figure 14.10 for a sequence of 7 instructions, in which the third instruction is a branch that is taken and in which there are no data dependencies.

	1	2	3	4	5	6	7	8	9	10
I1	FI	DA	FO	EX						
I2		FI	DA	FO	EX					
I3			FI	DA	FO	EX				
I4				FI	DA	FO				
I5					FI	DA				
I6						FI				
I7							FI	DA	FO	EX

14.11 Consider an instruction sequence of length n that is streaming through the instruction pipeline. Let p be the probability of encountering a conditional or unconditional branch instruction, and let q be the probability that execution of a branch instruction I causes a jump to a nonconsecutive address. Assume that each such jump requires the pipeline to be cleared, destroying all ongoing instruction processing, when I emerges from the last stage. Revise Equations (14.1) and (14.2) to take these probabilities into account.

第一条指令一定执行，需要时间 kτ，之后指令的执行都在第一条指令之后。

遇到跳转且跳转地址不连续的指令概率为：pq ，每条指令在流水线中所需时间kτ，因此：

剩余（n-1）条指令中跳转到不连续地址的指令所需时间 $pq(n-1)k\tau$ ；

其他指令概率为：1-pq ，每条指令在流水线中所需时间 τ， 因此：

剩余（n-1）条指令所需时间 $(1-pq)(n-1)\tau$ 。

$$\mathrm{Tk,n} = [\mathrm{k} + (pqk + (1-pq)) * (\mathrm{n}-1)]\tau$$

$$\mathrm{Sk} = \frac{\mathrm{T1,n}}{\mathrm{Tk,n}} = \frac{nk\tau}{[\mathrm{k}+(pqk+(1-pq))*(\mathrm{n}-1)]\tau} = \frac{nk}{(1-pq)[k+(n-1)]+pqnk}$$

Review Questions：15.1, 15.4, 15.5

15.1 What are some typical distinguishing characteristics of RISC organization?

(1). Large number of general purpose registers；

(2). Or use of compiler technology to optimize register use；

(3). Limited and simple instruction set；

(4). Emphasis on optimizing instruction pipeline.

15.4 What are some typical characteristics of a RISC instruction set architecture?

(1). One machine instruction per machine cycle；

(2). Register-to-register operations；

(3). Few, simple addressing modes；

(4). Few, simple instruction formats.

15.5 What is a delayed branch?

A way of increasing the efficiency of the pipeline, makes use of a branch that does not take effect until after execution of the following instruction.

Problems：15.2，15.10

15.2 In the discussion of Figure 15.2, it was stated that only the first two portions of a window are saved or restored. Why is it not necessary to save the temporary registers?

The temporary registers of level J are the parameter registers of level J + 1. Hence, those registers are saved and restored as part of the window for J + 1.

15.10 A SPARC implementation has *K* register windows. What is the number *N* of physical registers?

SPARC：1 window = 3 * 8 registers

physical registers = global registers + local registers

$N = 8 + K * （24 - 8）= 8 + 16K$

Problems：15.4，15.6

15.4 Reorganize the code sequence in Figure 15.6d to reduce the number of NOOPs.

Load rA←M	I	E_1	E_2	D					
Load rB←M		I	E_1	E_2	D				
NOOP			I	E_1	E_2				
Branch X				I	E_1	E_2			
ADD rC←rA+rB					I	E_1	E_2		
Store M←rC						I	E_1	E_2	D

15.6 Consider the following loop:

```
S: =0;
for K: =1 to 100 do
S: =S -K;
```

A straightforward translation of this into a generic assembly language would look something like this:

```
        LD   R1,0; keep value of S in R1
        LD   R2,1; keep value of K in R2
LP      SUB R1, R1, R2; S: =S-K
        BEQ R2,100, EXIT; done if K = 100
        ADD R2, R2, 1; else increment K
        JMP LP; back to start of loop
```

A compiler for a RISC machine will introduce delay slots into this code so that the processor can employ the delayed branch mechanism. The JMP instruction is easy to deal with, because this instruction is always followed by the SUB instruction: therefore, we can simply place a copy of the SUB instruction in the delay slot after the JMP, The BEQ presents a difficulty. We can't leave the code as is, because the ADD instruction would then be executed one too many times. Therefore, a NOP instruction is needed. Show the resulting code.

Solution 1:（按题目提示解）

```
        LD   R1,0
        LD   R2,1
        SUB R1, R1, R2
LP      BEQ R2,100, EXIT
        NOOP
        ADD R2, R2, 1
        JMP LP
        SUB R1, R1, R2
```

Solution 2:（更简洁）

```
        LD   R1,0
        LD   R2,1
LP      BEQ R2,100, EXIT
        SUB R1, R1, R2
        JMP LP
        ADD R2, R2, 1
```

Review Questions：16.4, 16.7

16.4 Briefly define the following terms:

True data dependency: A second instruction needs data produced by the first instruction.
Procedural dependency: The instructions following a branch (taken or not taken) have a procedural dependency on the branch and cannot be executed until the branch is executed.

Resource conflicts: A resource conflict is a competition of two or more instructions for the same resource at the same time.
Output dependency: Two instructions update the same register, so the later instruction must update later.

Antidependency: A second instruction destroys a value that the first instruction uses

16.7 What is the purpose of an instruction window?

For an out-of-order issue policy, the instruction window is a buffer that holds decoded instructions. These may be issued from the instruction window in the most convenient order.

16.3

- Consider the following assembly language program

  I1: Move R3,R7 /R3 ←(R7)/

  I2: Load R8,(R3) /R8←Memory(R3)/

  I3: Add R3,R3,4 /R3←(R3)+ 4/

  I4: Load R9,(R3) /R9 ← Memory (R3)/

  I5: BLE R8,R9,L3 /Branch if(R9)>(R8)/

  This program includes WAW, RAW, and WAR dependencies. Show these.

  WAW:I1-I3 RAW:I1-I2,I1-I3,I3-I4,I4-15,I2-I5 WAR:I2-I3

16.3

I1: Move R3,R7　　/R3 ←(R7)/

I2: Load R8,(R3)　/R8←Memory(R3)/

I3: Add R3,R3,4　　/R3←(R3)+ 4/

I4: Load R9,(R3)　/R9 ← Memory (R3)/

I5: BLE R8,R9,L3　/Branch if(R9)>(R8)/

分析相关性，按照指令顺序逐条分析

WAW：

1. I1 write R3　　寻找后面的指令是否又写了寄存器R3

   I2 write R8;I3 write R3;I4 write R9;I5 write none □ I1-I3
2. I2 write R8　　后面的指令没有写R8
3. I3 write R3　　后面的指令没有写R3
4. I4 write R9　　后面的指令没有写R9

16.3

I1: Move R3,R7 /R3 ←(R7)/

I2: Load R8,(R3) /R8←Memory(R3)/

I3: Add R3,R3,4 /R3←(R3)+ 4/

I4: Load R9,(R3) /R9 ← Memory (R3)/

I5: BLE R8,R9,L3 /Branch if(R9)>(R8)/

I1: Move R3a,R7a

I2: Load R8a,(R3a)

I3: Add R3b,R3a,4

I4: Load R9a,(R3b)

I5: BLE R8a,R9a,L3

RAW：

1. I1 write R3a 寻找后面的指令是否又读了寄存器R3a

I2 read R3a;I3 read R3a;I4 read R3b;I5 read R8a&R9a

□ I1-I2; I1-I3

2. I2 write R8a I5 read R8a&R9a □ I2-I5
3. I3 write R3b I4 read R3b □ I3-I4
4. I4 write R9a I5 read R8a&R9a □ I4-I5

16.3

I1: Move R3,R7 /R3 ←(R7)/

I2: Load R8,(R3) /R8←Memory(R3)/

I3: Add R3,R3,4 /R3←(R3)+ 4/

I4: Load R9,(R3) /R9 ← Memory (R3)/

I5: BLE R8,R9,L3 /Branch if(R9)>(R8)/

WAR:

1. I1 read R7 寻找后面的指令是否又写了寄存器R7

   I2 write R8;I3 write R3;I4 write R9;I5 write none

2. I2 read R3 I3 write R3 □ I2-I3
3. I3 read R3 none
4. I4 read R3 none

16.4

- a.Identify the RAW,WAW and WAR following dependencies in the instruction sequence:

```
I1:R1 = 100
I2:R1 = R2 + R4
I3:R2 = R4 - 25
I4:R4 = R1 + R3
I5:R1 = R1 + 30
```

RAW: I2&I4, I2&I5

WAR: I2&I3, I2&I4, I3&I4, I4&I5

WAW: I1&I2, I1&I5, I2&I5

- b.Rename the registers from part(a) to prevent dependency problems.Identify references to initial register values using the subscript “a" to the register reference.

16.4

- a.Identify the RAW,WAW and WAR following dependencies in the instruction sequence:

I1:R1 = 100	I1:R1a = 100
I2:R1 = R2 + R4	I2:R1b = R2a + R4a
I3:R2 = R4 - 25	I3:R2b = R4a - 25
I4:R4 = R1 + R3	I4:R4b = R1b + R3a
I5:R1 = R1 + 30	I5:R1c = R1b + 30

- b.Rename the registers from part(a) to prevent dependency problems.Identify references to initial register values using the subscript “a" to the register reference.

- Consider the "in-order-issue/in-order-completion" execution sequence shown in Figure 16.14.

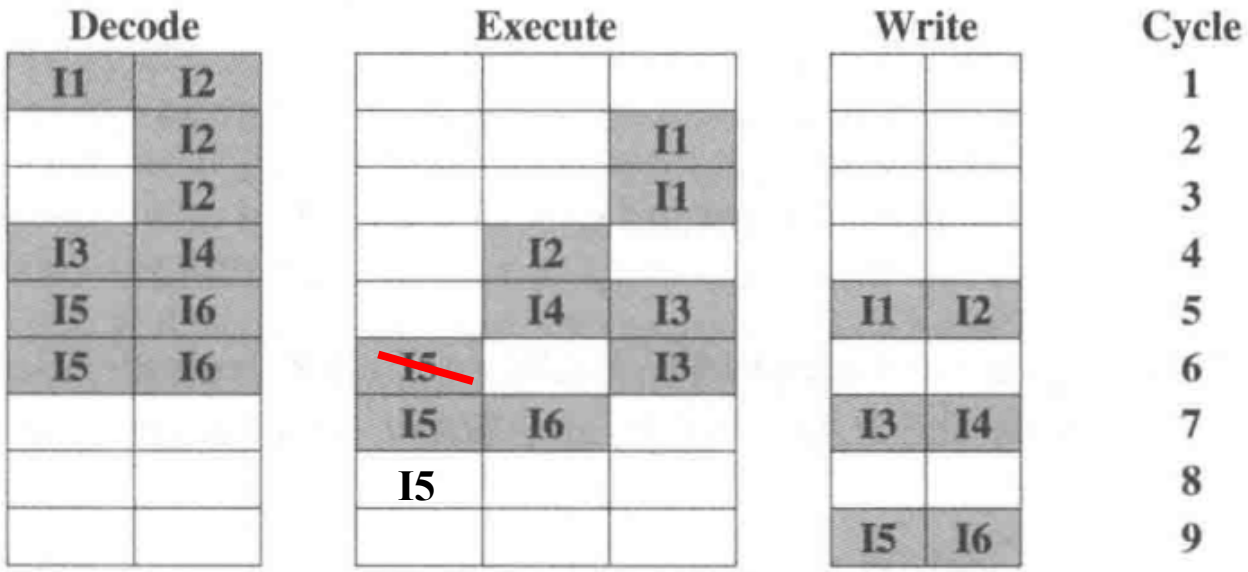


Figure 16.14 An In-Order Issue, In-Order-Completion Execution Sequence

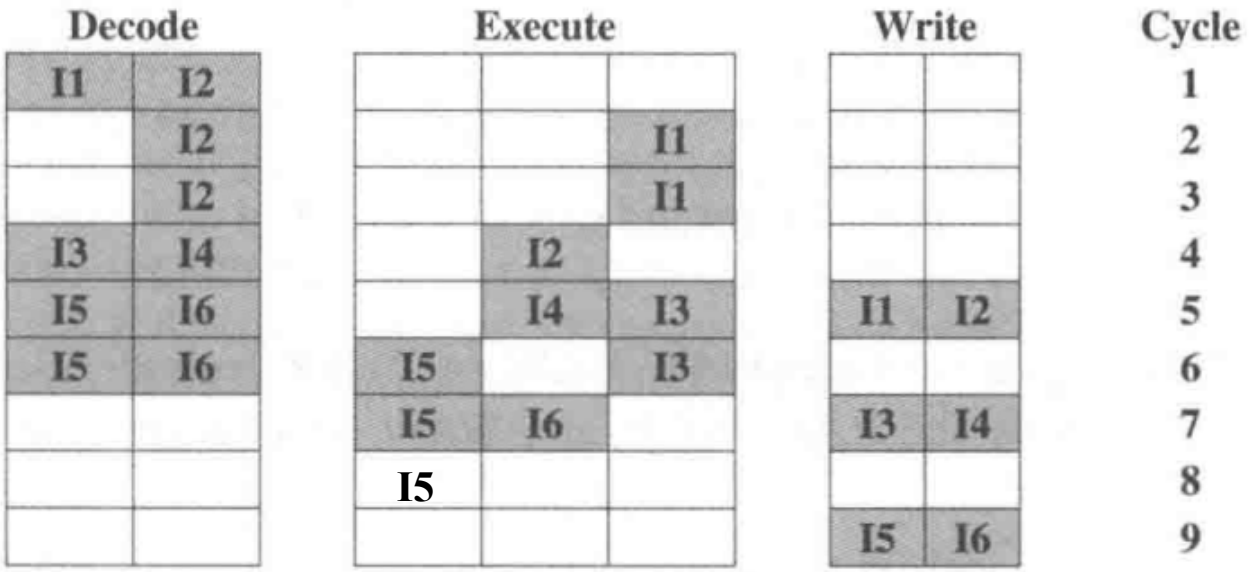


Figure 16.14 An In-Order Issue, In-Order-Completion Execution Sequence

I1 和 I2 在执行阶段使用的不是同一个资源，不是资源冲突；

I2在I1之后执行的原因可能是I2需要用到I1的结果；

真实数据相关无法被消除

16.5

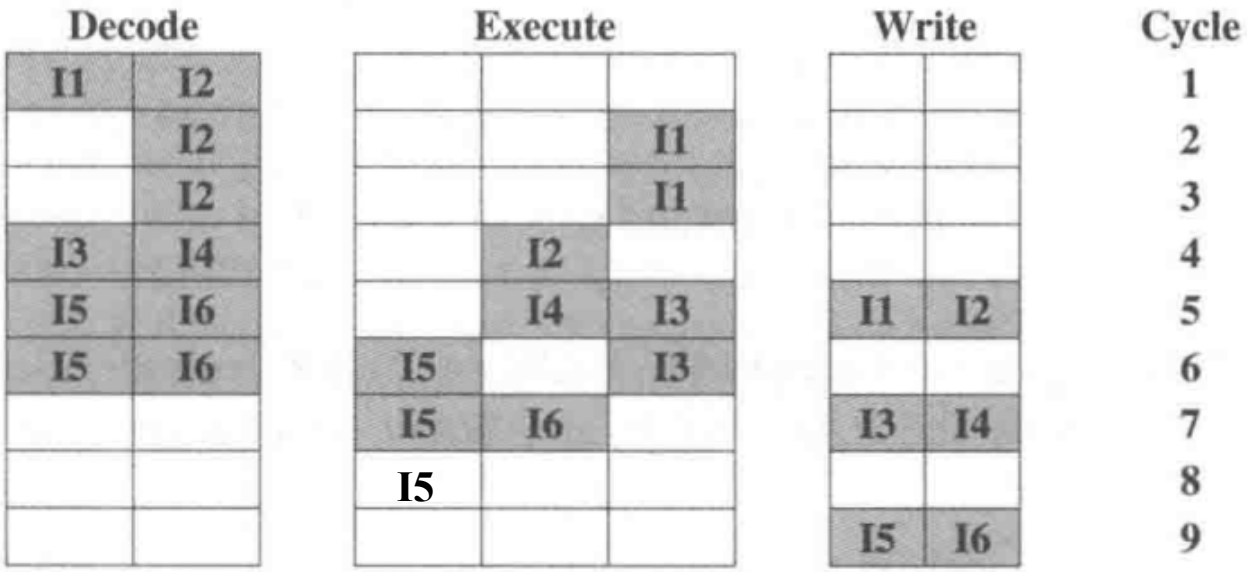


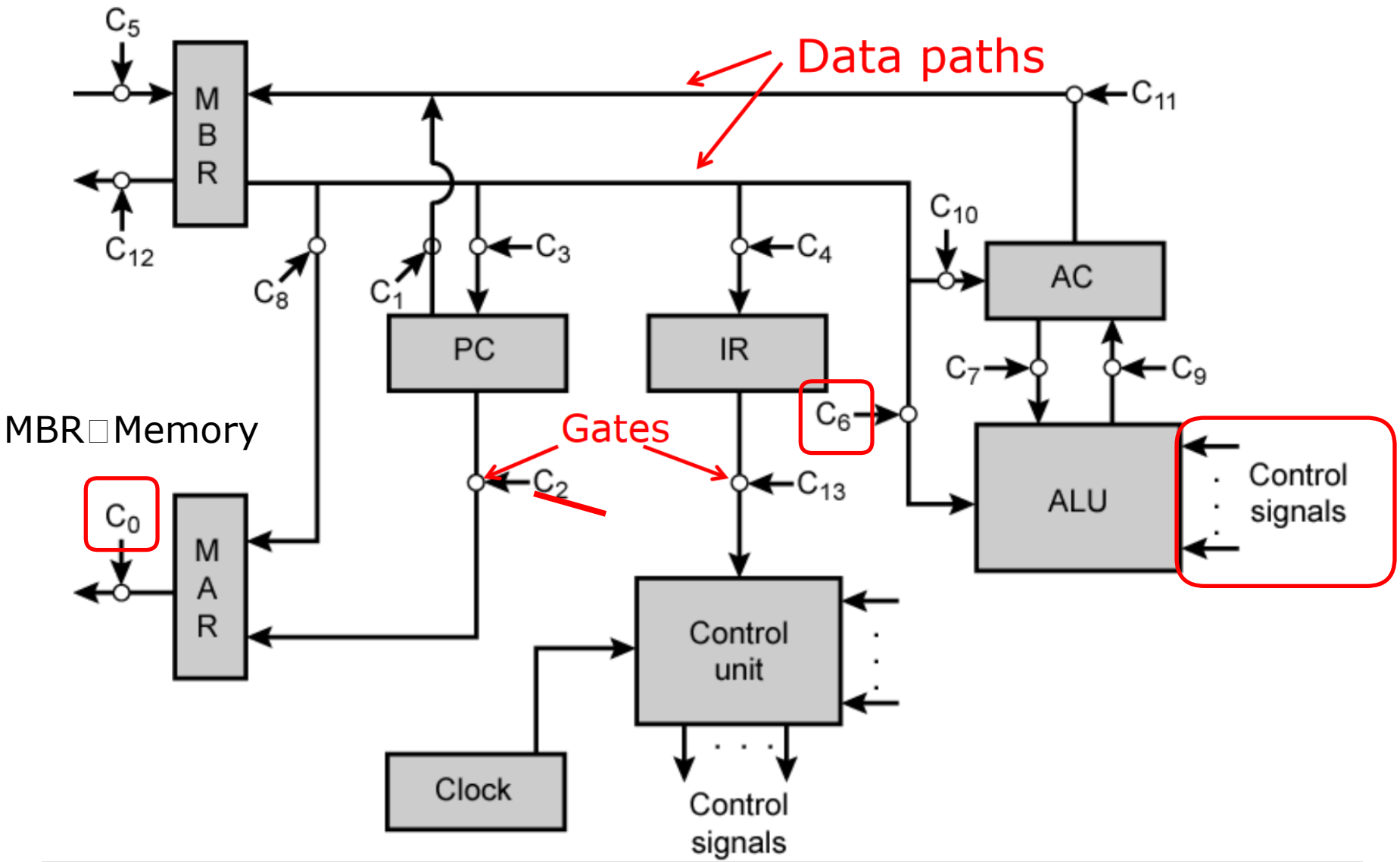
Figure 16.14 An In-Order Issue, In-Order-Completion Execution Sequence

I6 和 I5 同时译码，同时取指执行，说明不存在资源冲突个数据相关；

因为采用的是按序发射-按需完成，所以I6不能先于I5完成；

Out-of-completion 可以消除

20.2



MBR□Memory

20.2

Load AC:
t1:MAR□(IR(address)) C8
t2:MBR□ memory C5,CR,C0
t3:AC□(MBR) C10

Store AC
t1:MAR□(IR(address)) C8
t2:MBR□ (AC) C11
t3:memory □(MBR) C12,CW,C0

Add to AC:
t1:MAR□(IR(address)) C8
t2:MBR□ memory C5,CR,C0
t3:AC□ (AC)+(MBR) C6,C7,C9,CALU

AND:
t1:MAR□(IR(address)) C8
t2:MBR□ memory C5,CR,C0

20.2

JUMP:
t1:PC□(IR(address)) C3

JUMP if AC=0:
t1:test AC
  if AC = 0 , C3 is activated

Complement AC:
t1:AC□ (AC) C7,C9,CALU

20.4（假设已经取到指令）

a:
t1:Y□(IR(address))
t2:Z□ (AC)+(Y)
t3:AC□(Z)

b:
t1:MAR□(IR(address))
t2:MBR□ memory
t3:Y □(MBR)
t4:Z□ (AC)+(Y)
t5:AC□(Z)

c:
t1:MAR□(IR(address))
t2:MBR□ memory
t3:MAR □(MBR)
t4:MBR□ memory
t5:Y □(MBR)
t6:Z□ (AC)+(Y)
t7:AC□(Z)

20.5

- A stack is implemented as shown in Figure 20.11. Show the sequence of micro-operations for
  a. popping
  b. pushing the stack

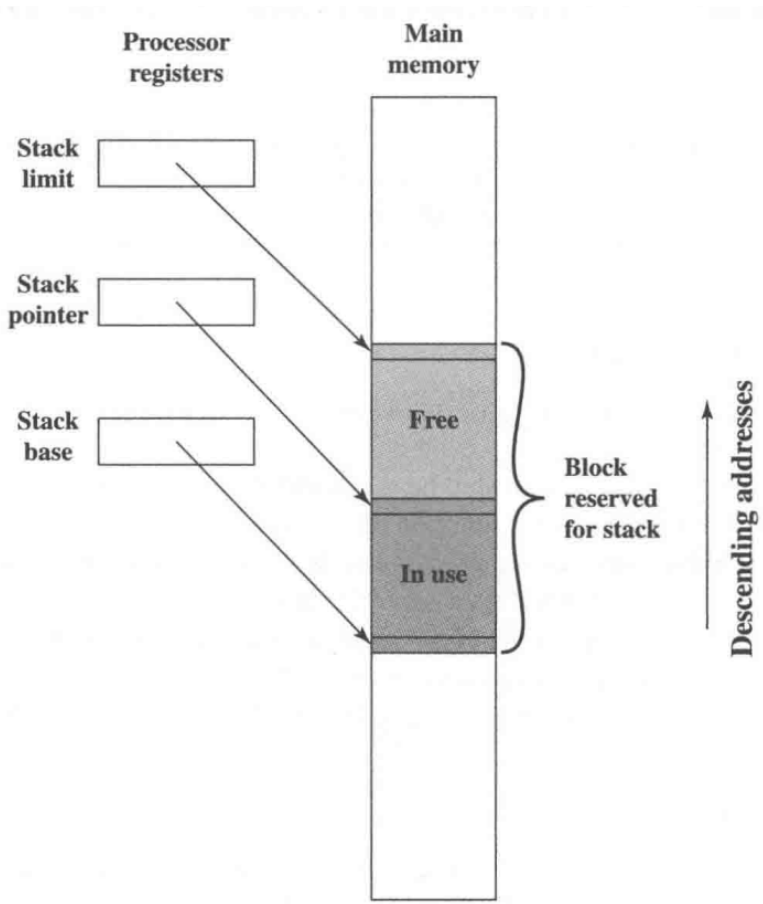


Figure 20.11 Typical Stack Organization (full/descending)

20.5

- a. popping

➢ POP

SP□(SP) + 1

➢ POP $\overline{X}$

t1: MAR□(SP)

t2: MBR□Memory

SP□(SP)+1

t3: MAR□(IRaddress)

t4: Memory□(MBR)

➢ POP R1

t1:MAR□(SP)

t2:MBR□Memory

SP□(SP)+1

t3:R1□(MBR)

- b. pushing the stack

➢ PUSH X

t1: SP □(SP)-1

MAR□(IRaddress)

t2: MBR□Memory

t3: MAR□ (SP)

t4: Memory□(MBR)

➢ PUSH R1

t1: SP □(SP)-1

t2: MBR□(R1)

MAR□ (SP)

t3: Memory□(MBR)

Problems：16.3，16.4，16.5

21.3.

a. These flags represent Boolean variables that are input to the control unit logic. Together with the time input and other flags, they determine control unit output.

b. The phase of the cycle is implicit in the organization of the microprogram. Certain locations in the microprogram memory correspond to each of the four phases.

21.4

code	address field	jump conditions
13	8	3

a. 8flag → 2^3 → 3 bit

b.24-13-8 = 8bit

c. 2^8 * 24bit = 6144 bits

21.6

1024words →2^{10} →address field : 10bits

opcode : 5bits

不同的opcode对应machine instruction routine，之间相差8条微指令→至少3bits

前面2bit任意，如：

00 ***** 000/ 01*****000

或者0*****0000…

21.7

题干要求分为子域来控制

分为两个域a&b

a=0时，b决定微操作；b=0时，a决定微操作

a + b = 9

2^a -1 + 2^b -1 = 46

解得：a或b分别为4或5bit