Assignment 2

3190102214 张沛全

01

a.

1. Data-level parallelism: operate on multiple available data items at the same time
2. Task-level parallelism: create multiple tasks of work that can operate independently and largely in parallel

b.

1. Instruction-level parallelism: exploits data-level parallelism at modest levels with compiler help using ideas like pipelining and at medium levels using ideas like speculative execution.
2. Vector architectures, graphic processor units (GPUs), and multimedia instruction sets: exploit data-level parallelism by applying a single instruction to a collection of data in parallel.
3. Thread-level parallelism: exploits either data-level parallelism or task-level parallelism in a tightly coupled hardware model that allows for interaction between parallel threads.
4. Request-level parallelism: exploits parallelism among largely decoupled tasks specified by the programmer or the operating system.

02

$\text{Energy}_{\text{dynamic}} \propto \text{Capacitive load} \times \text{Voltage}^2$

$\text{Power}_{\text{dynamic}} \propto 1/2 \times \text{Capacitive load} \times \text{Voltage}^2 \times \text{Frequency}$

dynamic energy: $\frac{\text{newEnergy}}{\text{oldEnergy}} = \frac{(1-0.15)^2}{1^2} = 0.7225$

dynamic power: $\frac{\text{newPower}}{\text{oldPower}} = \frac{(1-0.15)^2 \times (1-0.15)}{1^2 \times 1} = 0.61$

Dynamic energy becomes 72.25% of the original and dynamic power becomes 61% of the original.

03

a.

There is no need for a microprocessor to always operate at the highest clock frequency and voltages, especially when there are periods of low activity. Modern microprocessors typically offer a few clock frequencies and voltages in which to operate that use lower power and energy. To adjust clock frequency and voltages can save about 10%-15% power in general.

b.

The chip can decide that it is safe to run at a higher clock rate for a short time, possibly on just a few cores, until temperature starts to rise, which can be influenced by the room temperature. Overclocking helps to work faster. Since overclocking works only when the temperature allows, it can improve energy efficiency.

04

a.

Failure rate
= $10 \times \frac{1}{1{,}000{,}000} + \frac{1}{500{,}000} + \frac{1}{200{,}000} + \frac{1}{200{,}000} + \frac{1}{1{,}000{,}000}$
=$\frac{23}{1{,}000{,}000}$

MTTF$_{\text{system}}$ = $\frac{1}{\text{Failure rate}} \approx 43478$ hours

b.

According to the characteristic of exponential distribution, Failure rate$_1$
= $10 \times \frac{1}{1{,}000{,}000} + \frac{1}{500{,}000} + \frac{1}{1.5 \times 200{,}000} + \frac{1}{200{,}000} + \frac{1}{1{,}000{,}000}$
=$\frac{64}{3{,}000{,}000{,}000}$

MTTF$_{\text{system1}}$ = $\frac{1}{\text{Failure rate}_1} = 46875$ hours

05

a.

execution time$_{\text{new}}$ = execution time$_{\text{old}}$ $\times$ ((1-Fraction$_{\text{enhanced}}$) + Fraction$_{\text{enhanced}}$ / Speedup$_{\text{enhanced}}$)

b.

No. The enhancement fraction includes those operations without dependencies, which can be parallelly executed. Other operations can only be serially executed, which cannot be optimized more. So the fraction allowing parallelism decides the ultimate performance improvement.

06

Original CPI: $4 \times 25\% + 1.33 \times 75\% = 2$

CPI of design 1: $2 - 2\% \times (20 - 2) = 1.64$

CPI of design 2: $2 - 25\% \times (4 - 2) = 1.5$

So the second design is better

07

a.

Energy = Power X Time

So 50% of the energy is saved.

b.

Energy = $\frac{1}{2} \times$ Capacitive load $\times$ Voltage^2

So the energy is reduced to 25% of the original. That is, 75% of the energy is saved.

08

a.

Execution$_{\text{new}}$ = $0.6 + \frac{0.4}{2} = 0.8$

Speedup = $\frac{1}{0.8} = 1.25$

b.

Execution$_{\text{new}}$ = $0.01 + \frac{0.99}{2} = 0.505$

Speedup = $\frac{1}{0.505} = 1.98$

c.

Execution$_{\text{new}}$ = $(0.6 + \frac{0.4}{2}) \times 0.8 + 0.2 = 0.84$

Speedup = $\frac{1}{0.84} = 1.19$

d.

Execution$_{\text{new}}$ = $0.8 + 0.2 \times (0.01 + \frac{0.99}{2}) = 0.901$

Speedup = $\frac{1}{0.901} = 1.11$

09

a.

Execution$_{\text{new}}$ = $0.2 + \frac{0.8}{N}$

Speedup = $\frac{1}{0.2+0.8/N} = \frac{N}{0.2N+0.8}$

b.

Execution$_{\text{new}}$ = $0.2 + \frac{0.8}{8} + 8 \times 0.5\% = 0.34$

Speedup = $\frac{1}{0.34} = 2.94$

c.

Execution$_{\text{new}}$ = $0.2 + \frac{0.8}{8} + \log_2 8 \times 0.5\% = 0.315$

Speedup = $\frac{1}{0.315} = 3.17$

d.

Execution$_{\text{new}}$ = $0.2 + \frac{0.8}{N} + \log_2 N \times 0.5\%$

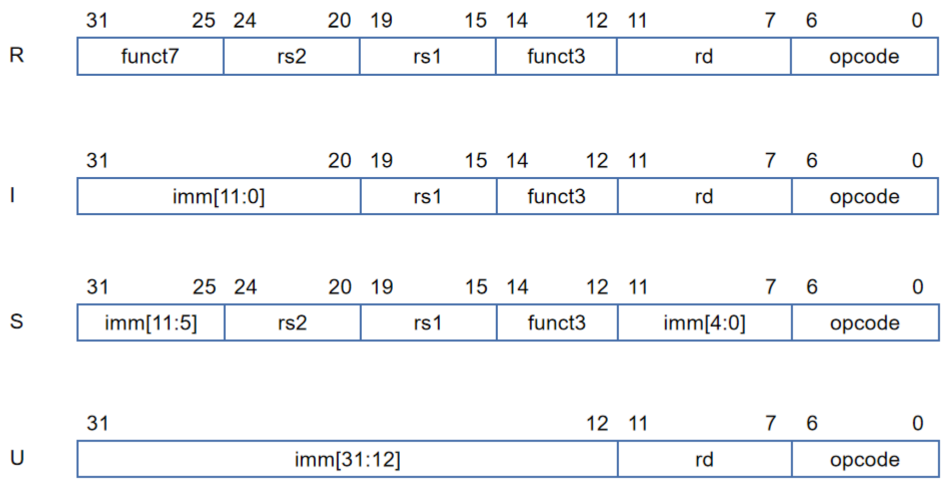
Speedup = $\frac{1}{0.2+\frac{0.8}{N}+\log_2 N \times 0.5\%}$

e.

$$\frac{d\, 1/(1-P+\frac{P}{N}+\log_2 N \times 0.5\%)}{dN} = 0$$

10

a.



b.

- funct7: 7-bit function code(additional opcode)
- rs2: the second source register number
- rs1: the first source register number
- funct3: 3-bit function code(additional opcode)
- rd: destination register number
- opcode: operation code
- imm[xx:xx]: bits making up an immediate

c.

0x00000013, which can be interpreted as: 0000000 00000 00000 000 00000 0010011. The opcode, funct3 and funct7 together tells that this instruction is the addition between registers. We can know that rd, rs1 and rs2 are all x0. Combining aforementioned information, we can know the instruction is `add x0, x0, x0`

11

a. Immediate

The operand is a constant within the instruction itself e.g `addi s1, s1, 0`

b. Displacement

The operand is at the memory location whose address is the sum of a register and a constant in the instruction e.g `ld s1, 8(t1)`

c. Register indirect

The operand is at the memory location whose address is the value of a register e.g `mov ax, ds:[bx]` in x86

d. Direct or absolute

The operand is at the memory location whose address is given within the instruction itself e.g `mov ax, ds:[0x888888]` in x86

e. Memory indirect

The operand is at the memory location whose address is given by a pointer at a memory location whose address is the value of a register e.g `add r1, @(r3)`

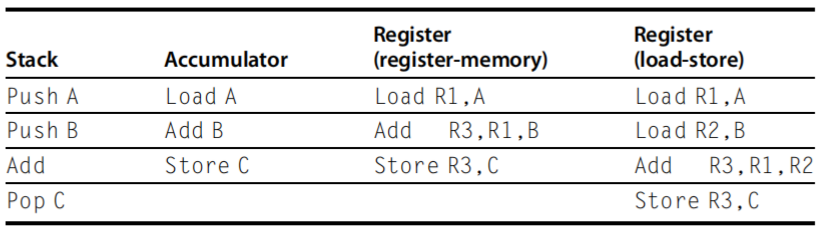
f. Scaled

The operand whose address is the sum of a register or a constant and a register multiplied by the size of an element, which is used to index arrays, e.g `mov ax, 100H[si*2]` in x86

g. PC-relative or position independence

The operand (branch address) is the sum of the PC and a constant in the instruction e.g `beq x0, x0, 0x20`

12



Stack	Accumulator	Register (register-memory)	Register (load-store)
Push A	Load A	Load R1,A	Load R1,A
Push B	Add B	Add R3,R1,B	Load R2,B
Add	Store C	Store R3,C	Add R3,R1,R2
Pop C			Store R3,C

Figure A.2 The code sequence for C = A + B for four classes of instruction sets. Note that the Add instruction has implicit operands for stack and accumulator architectures and explicit operands for register architectures. It is assumed that A, B, and C all belong in memory and that the values of A and B cannot be destroyed. Figure A.1 shows the Add operation for each class of architecture.

stack:

- 3 memory addresses
- 3*64+4*8 = 224

accumulator:

- 3 memory address
- 3*(64+8) = 216

register (register-memory):

- 4 register addresses + 3 memory addresses
- 3*8 + 3*64 + 4*6 = 240

register (load-store):

- 6 register addresses + 3 memory addresses
- 4*8 + 3*64 + 6*6 = 260

13

a.

```
loop: li t0, 100        # bound
      li t1, 1000       # A
      li t2, 3000       # B
      li t3, 5000       # C
```

```
        li t4, 7000        # i
        ld s4, 0(t4)       # load i
        ld s3, 0(t3)       # load C
        slli t5, s4, 3
        add t2, t5, t2     # calculate address of B[i]
        ld s2, 0(t2)       # load B[i]
        add t1, t5, t1     # calculate address of A[i]
        add s1, s2, s3     # A[i] = B[i] + C
        sd s1, 0(t1)       # save A[i]
        addi s4, s4, 1     # i++
        sd s4, 0(t4)       # save new i
        ble s4, t0, loop   # i<=100
```

b.

5*101 = 505

c.

For RV32I, an `li` is equivalent to an `lui` and an `addi` if necessary. So the exact size is (16+3)*4 = 76 bytes

14

Clock cycles in A: $\frac{1}{4G} = 0.25$ ns

Clock cycles taken for the program in A: $\frac{10\ s}{0.25\ ns} = 4 \times 10^{10}$

Clock cycles taken for the program in B: $1.2 \times 4 \times 10^{10} = 4.8 \times 10^{10}$

Clock cycles in B: $\frac{6\ s}{4.8 \times 10^{10}} = 0.125\ ns$

Frequency of B: $\frac{1}{0.125\ ns} = 8G$ Hz

15

- I/O device request
- Invoking an operating system service from a user program
- Tracing instruction execution
- Breakpoint (programmer-requested interrupt)
- Integer arithmetic overflow
- FP arithmetic anomaly
- Page fault (not in main memory)
- Misaligned memory accesses (if alignment is required)
- Memory protection violation
- Using an undefined or unimplemented instruction
- Hardware malfunctions
- Power failure

16

a.

There are a separate memory and caches storing instructions and a separate memory and caches storing data to use. In phase IF and MEM, pipeline needs to access memory. If there is only one memory, these two phases should be serialized. With a two-memory technique, these two phases can be paralleled, which simplifies the design and improves efficiency.

b.

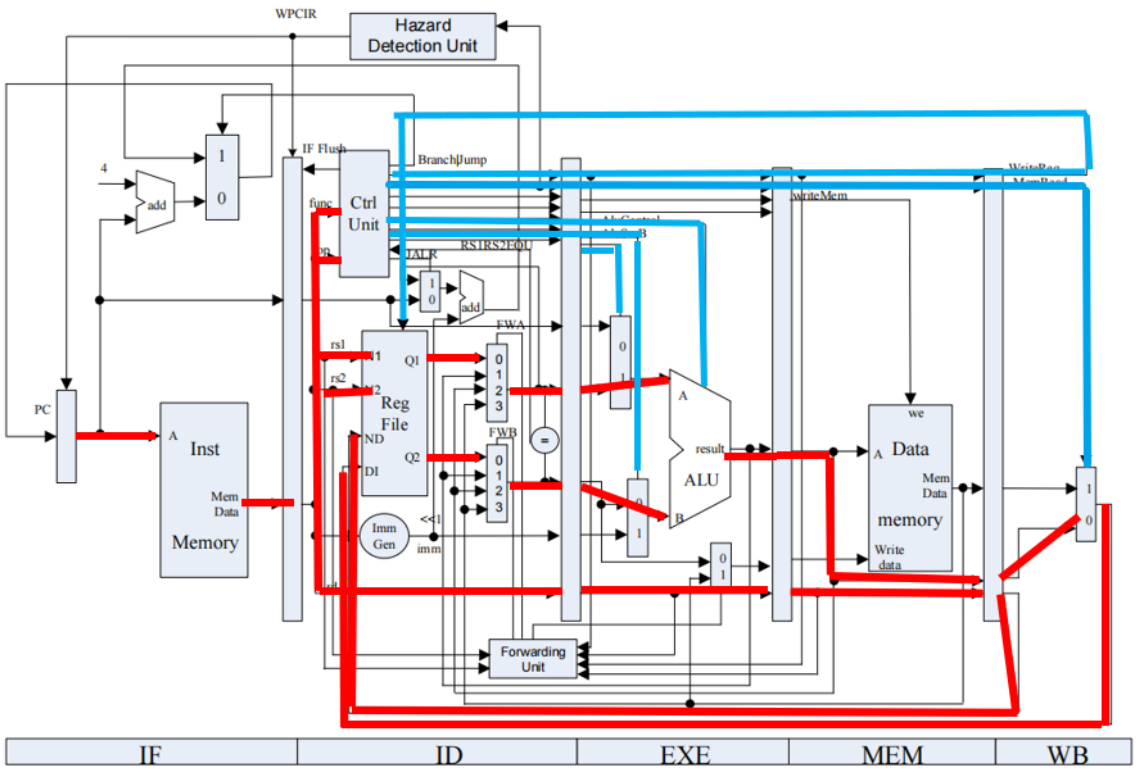
Phase IF and MEM can be paralleled and there are more stages in a pipeline. The performance is improved.

17

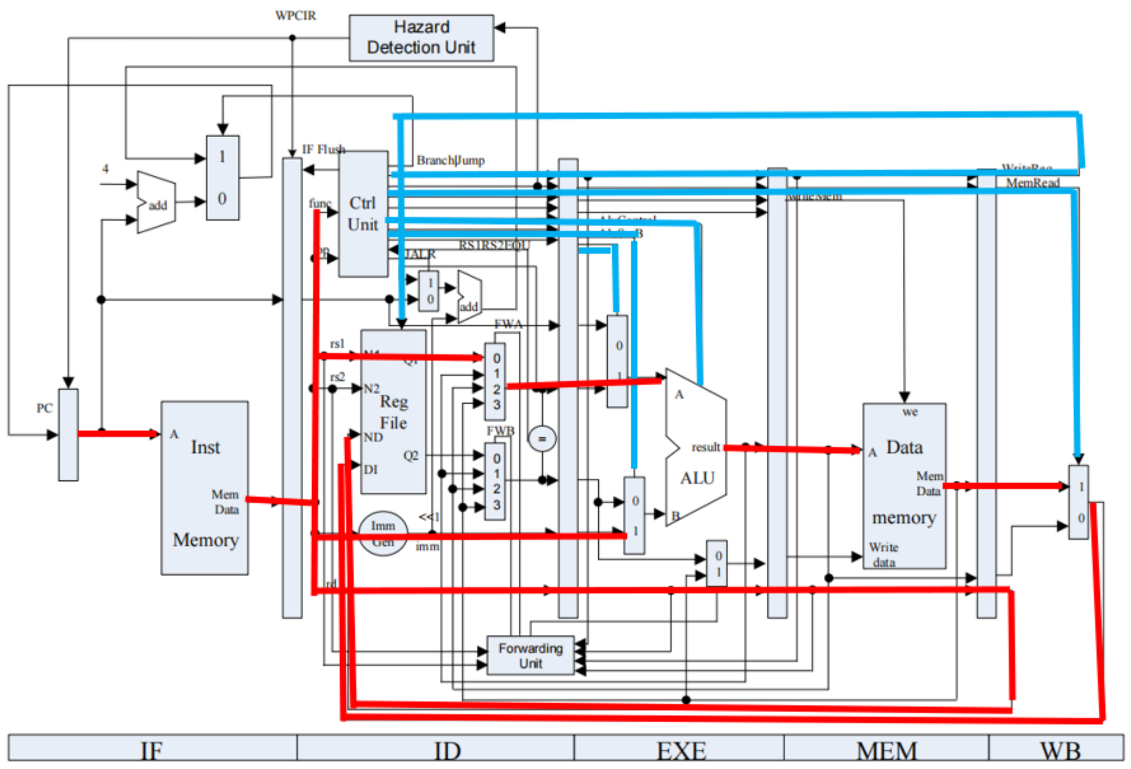
Because the length of a clock cycle is determined by the longest pipe stage. If pipe stages are not of equal lengths, time would be wasted on the longest pipe stage.

18

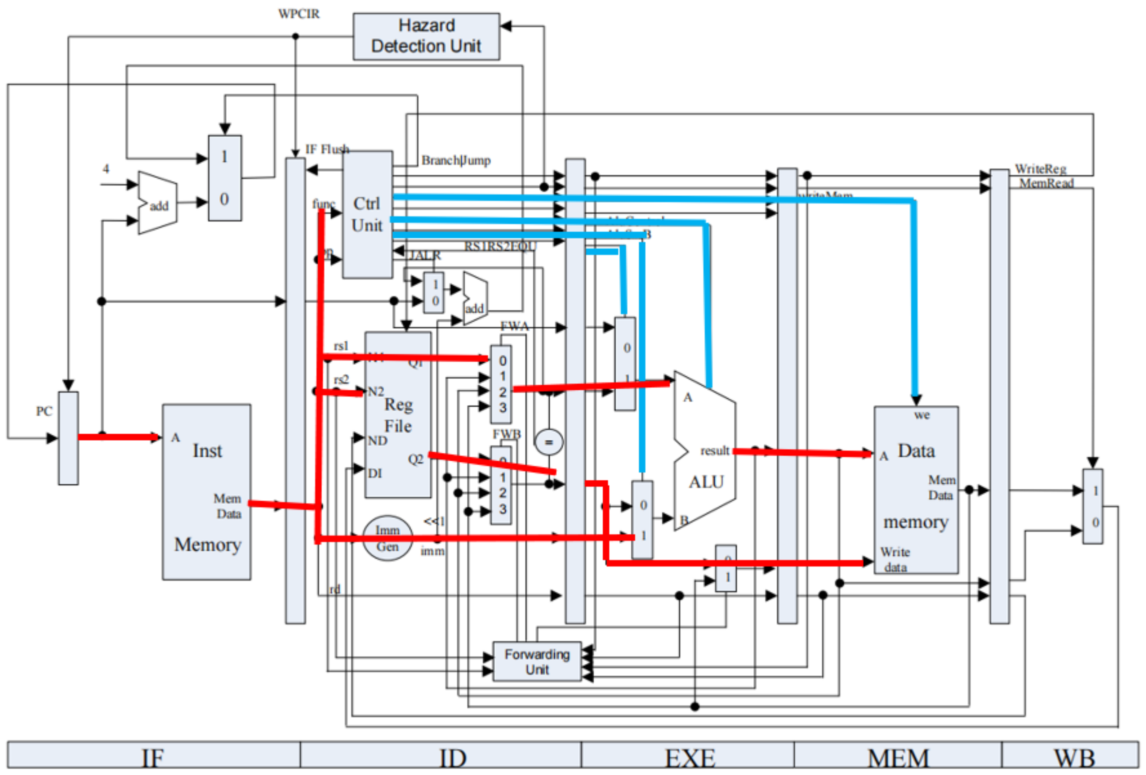
add



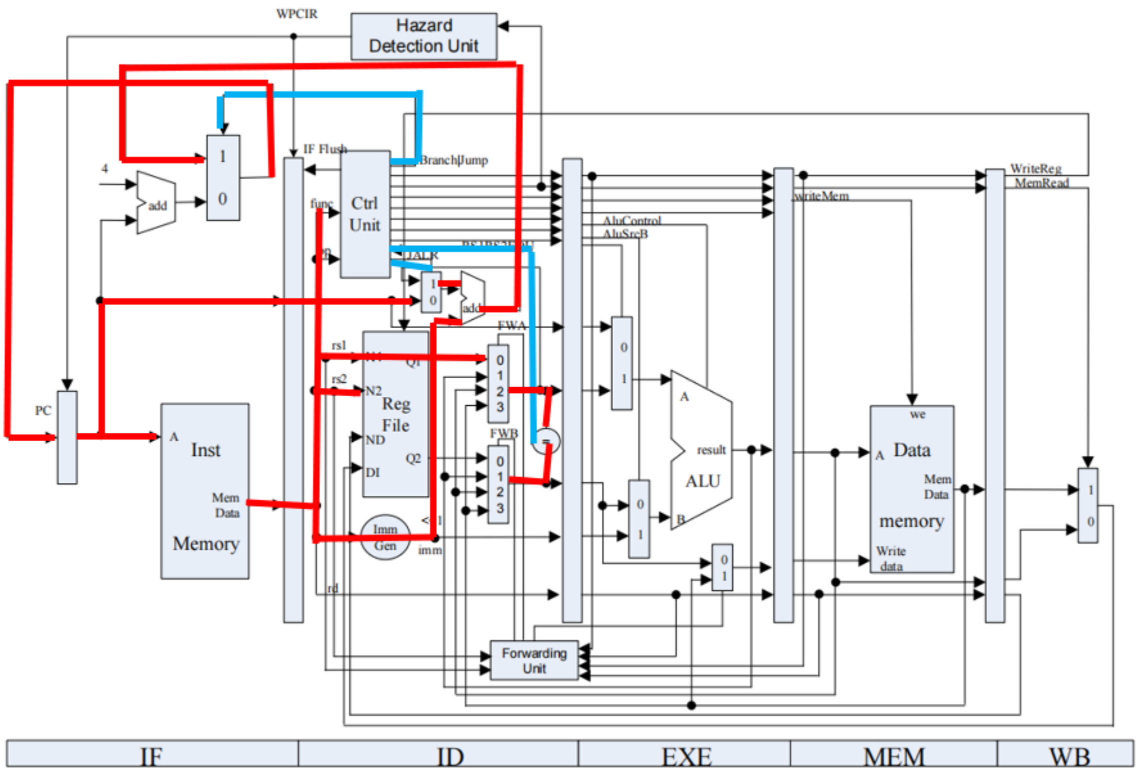
ld


WPCIR
Hazard Detection Unit
IF Flush
Branch|Jump
Ctrl Unit
func
op
JALR
RS1RS2EQU
ALUControl
ALUSrcB
WriteMem
WriteReg
MemRead
4
add
1
0
PC
A
Inst
Mem Data
Memory
rs1
rs2
N1
N2
ND
DI
Q1
Q2
Reg File
FWA
FWB
Imm Gen
<<1
imm
rd
ALU
result
B
we
Data
Mem Data
memory
Write data
Forwarding Unit
IF
ID
EXE
MEM
WB

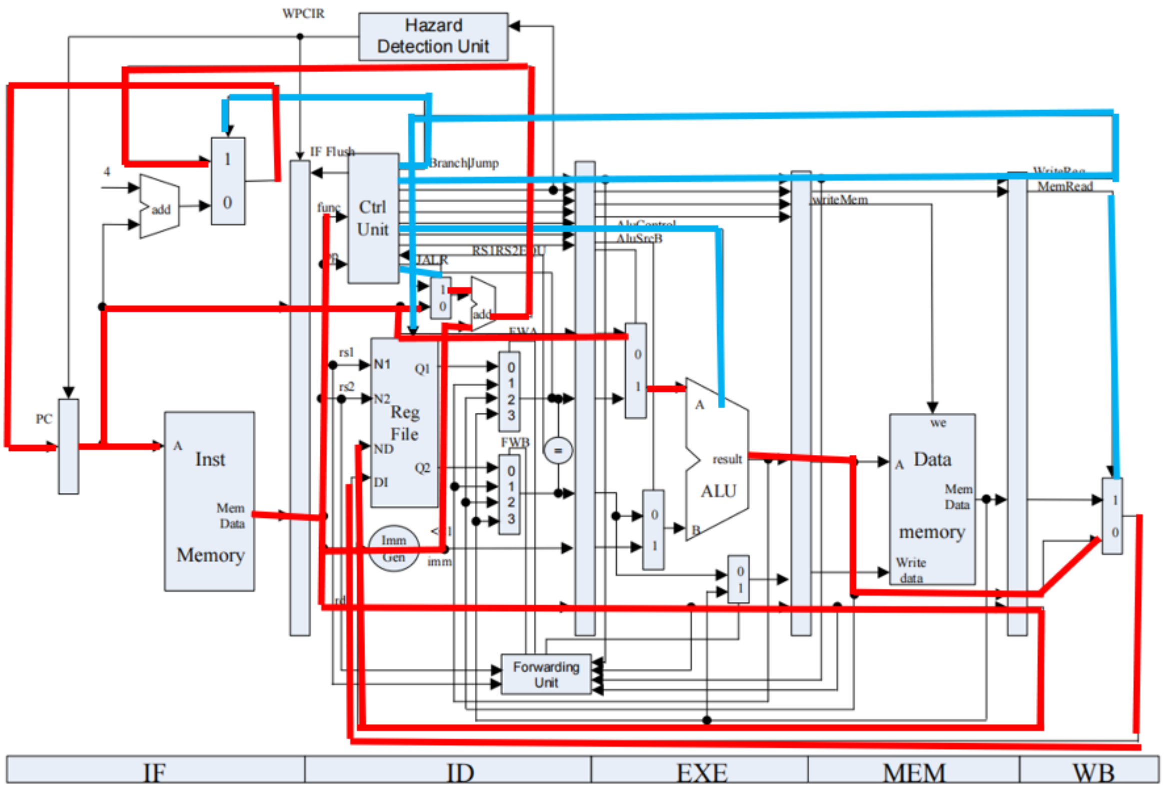
sd


WPCIR
Hazard Detection Unit
IF Flush
Branch|Jump
Ctrl Unit
func
op
JALR
RS1RS2EQU
ALUControl
ALUSrcB
WriteMem
WriteReg
MemRead
4
add
1
0
PC
A
Inst
Mem Data
Memory
rs1
rs2
N1
N2
ND
DI
Q1
Q2
Reg File
FWA
FWB
Imm Gen
<<1
imm
rd
ALU
result
B
we
Data
Mem Data
memory
Write data
Forwarding Unit
IF
ID
EXE
MEM
WB

beq



jal



19

- Without
  - `a[i][j] = b[j][0] * b[j+1][0]`: 100/2*3+101 = 251
- With
  - `a[0][j] = b[j][0] * b[j+1][0]`: 7+4 = 11

- `a[i][j] = b[j][0] * b[j+1][0]`: 4+4 = 8
- 19 in total

20

a.

Miss rate is 0.

b.

192: 100%

320: 100%

c.

64: 0 still, no benefit

192: miss rate is about (192-128)/192 = 33.3%

320: miss rate is about (320-128)/320 = 60%

21

Cycles from L2 to I-cache: $\lceil \frac{32\times8}{128} \times \frac{1.1}{0.266} \rceil = 9$

Cycles from L2 to D-cache: $\lceil \frac{16\times8}{128} \times \frac{1.1}{0.266} \rceil = 5$

Cycles from memory to L2: $\lceil \frac{64\times8}{128} \times \frac{1.1}{0.133} \rceil = 34$

Average cycles from memory to L2: $c_{M2L2} = 60 \times 1.1 + 34 = 100$

a. instruction access

$2\% \times (\lceil 15 \times 1.1 \rceil + 9 + 20\% \times c_{M2L2}) = 0.92$

$0.92 \times 1.1 = 1.01\ ns$

b. data reads

$5\% \times (\lceil 15 \times 1.1 \rceil + 9 + 20\% \times t_{M2L2}) = 2.3$

$2.3 \times 1.1 = 2.53\ ns$

c. data writes

There is no enough conditions in the problem description, so I suppose the penalty time includes the time for replacement and writing. Average memory access time for data writes is $2.07\ ns$.

d. overall CPI

$0.7 + 0.92 + (0.2 + 0.05) \times 2.3 = 2.195$

22

a.

cycles: 1 + 105*0.05 = 6.25

b.

hit rate: 64K / 256M = 0.00025

cycles: 1 + (1-0.00025)*105 = 105.97

c.

Assume the miss rate to be m

$m \times L \le (1-m) \times G$

So the miss rate $m \le 95\%$

23

a.

First, the 64-bit virtual address is logically divided into a 50-bit virtual page number and 14-bit page offset. The former is sent to the TLB to be translated into a physical address, and the high bit of the latter is sent to the L1 cache to act as an index. If the TLB match is a hit, then the 26-bit physical page number is sent to the L1 cache tag to check for a match. If it matches, it's an L1 cache hit. The 6-bit block offset then selects the word for the processor.

If the L1 cache check results in a miss, the 40-bit physical address is then used to try the L2 cache. The middle 14-bit portion of the physical address is used as an index to the 4 MiB L2 cache. The resulting L2 cache tag is compared with the upper part of the physical address to check for a match. If it matches, we have an L2 cache hit, and the data are sent to the processor, which uses the block offset to select the desired word. On an L2 miss, the physical address is then used to get the block from memory.

L1 is virtually indexed and physically tagged and L2 is physically indexed and physically tagged.

b.

If cache size is larger than page size, the bits needed to locate a byte in a cache will be more than the bits needed to locate a byte in a page. Some bits supposed to compose virtual page number will be used for that. Such a design is inefficient and difficult to implement.

bits: $page \ge index + offset$

cache size: $2^{index} \times 2^{offset} \le 2^{page} =$ page size

c.

If cache size is larger than n times of the page size, the bits needed to locate a set in a cache will be more than the bits of a virtual address used to represent L1 cache index. Some bits supposed to compose virtual page number will be used for that.

bits: $page \ge index + offset$

cache size: $2^{index} \times 2^{offset} \times n \le n \times 2^{page} =$ n times of the page size

24

offset:

- bits: $\log_2 16 + 1 = 5$ (words are addressed to half-word)

index:

- blocks: $\frac{2M}{16\times 8} = 2^{14}$
- bits: $\log_2 2^{14} - \log_2 16 = 10$

tag:

- bits: $64 - 5 - 10 = 49$

25

I really appreciate that I have met kg in my college life. kg is one of the few teachers I have met that make a connection with their students. I have learned a lot from kg, both in class and outside class. In class, I do appreciate kg’s efforts on teaching and interacting with students and I do agree that kg can elaborate key points without loss of simplicity. Questions raised in class always make me thinking. Outside class, being guided by kg in an SRTP project, I think I know a little more about kg’s scientific research and life. According to my observation, I share some characters with kg, maybe.

Back to the issue of future planning, in fact, I think myself still wandering in the world of computer science, though entering the second half of my undergraduate career. I think what I have mastered is basic and is still far from application, so in the future I would like to try to seek a higher academic degree. It is also an escape for me. Because I really hate 996 but I can hardly see hope for avoiding such life. If I pursue a master’s degree for the coming years, I could gain more competitiveness and live more freely for another 3 years. That sounds like a quite reasonable tradeoff. There are still some problems for this decision, I may work them out and find answers in the future. My mother told me do not worry too much about what hasn't happened yet.

When writing these lines, I reviewed what kg has written for us and was touched again. I can feel that kg’s zeal for teaching hasn’t faded away, though there are many times such zeal can vanish little by little in class, especially when kg tries to interact but no one responds. There are majorly 3 reasons in my eyes: some students’ English is not that good and thus are unable to catch up; some students just tend to be silent, like me; the setting of courses, that is, the contents of 3 hardware courses overlap seriously, in my opinion. However, there still exists some guys that respond to kg! So hope that kg’s zeal will never fade and wish kg good health.