MIPS Lecture 1

Liuli Chen

Mar 2, 2012

Course Overview

- What will you do?
  - simulate a MIPS CPU
    - Pipeline
    - Cache
    - Advance module (or Simics)
- What will you use?
  - Modelsim 6.5b SE
  - Verilog

TAs

- Diyi Yang（杨笛一）
- Hongyu Zhu（朱虹宇）
- Liuli Chen（陈旒俐）

Schedule

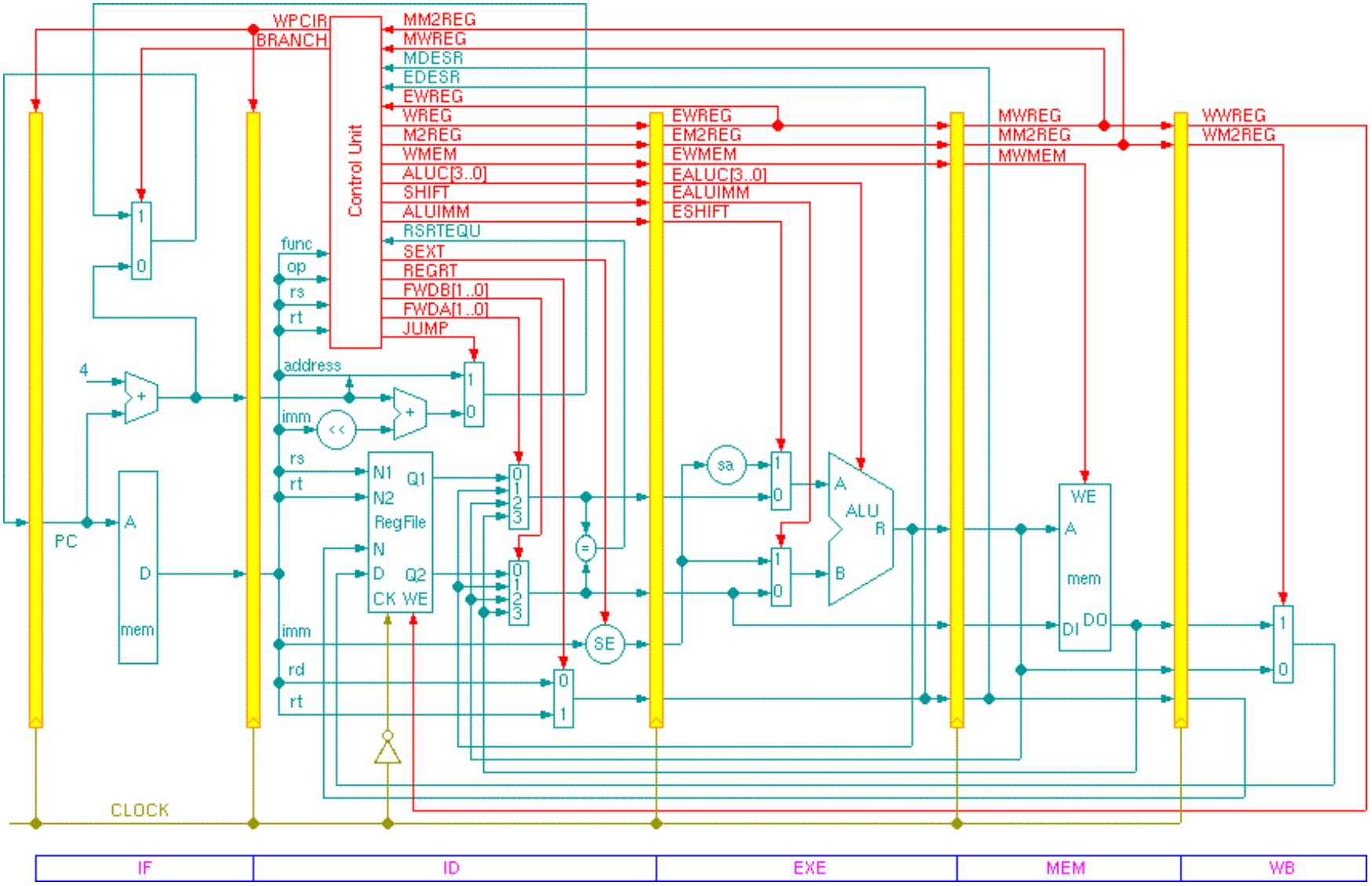
- 3.2: First lecture(Pipeline)
- 3.25(tentative): Deadline for Pipeline Phase && Second lecture(Cache)
- 4.16(tentative): Deadline for Cache Phase && Third lecture(Optional Module)
- 5.8(tentative): Final submit
- 5.15(tentative): Presentation

- **No Delay!**

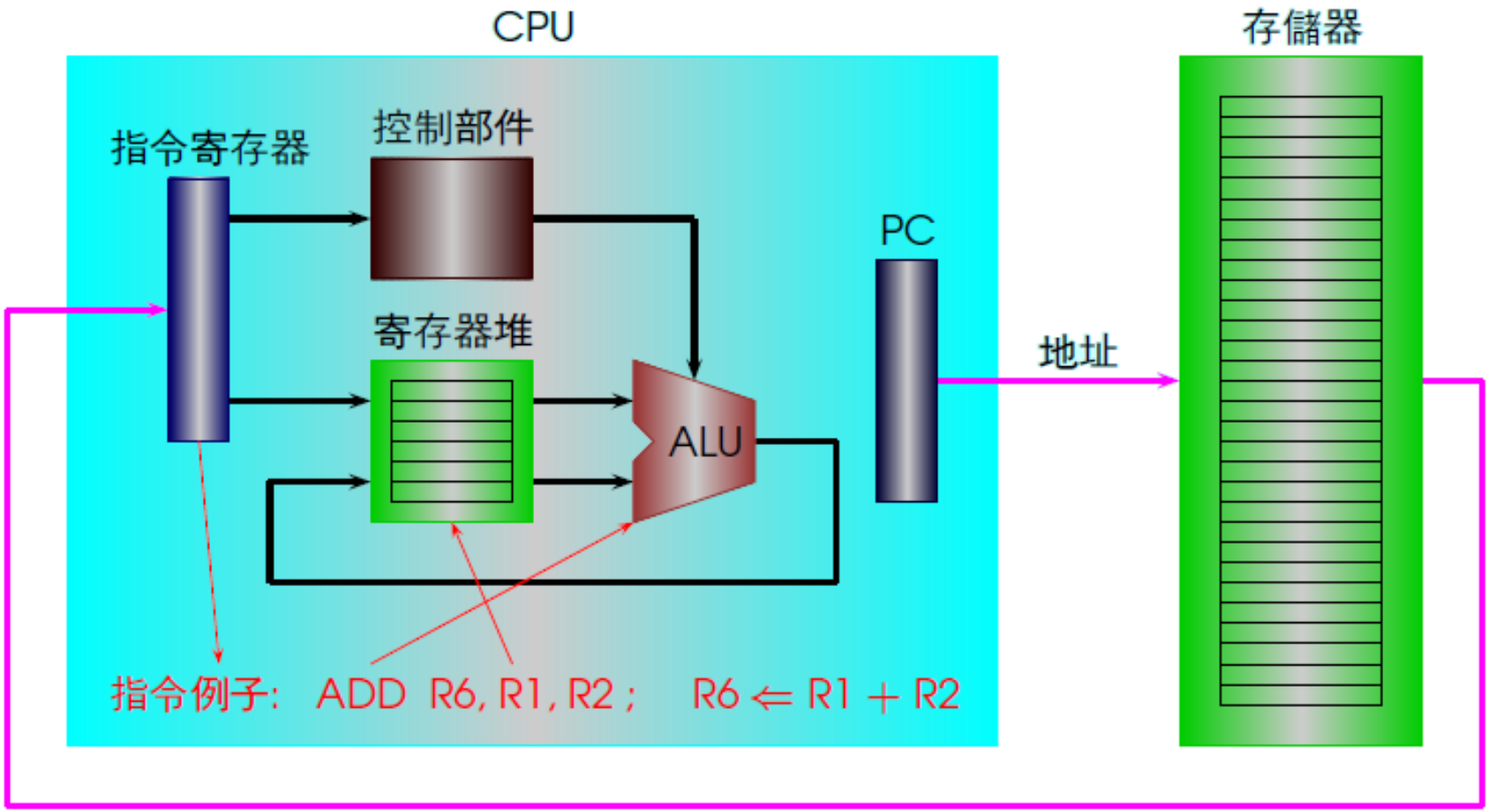
MIPS architecture

- What is MIPS?
  - Microprocessor without Interlocked Pipeline Stages
  - a reduced instruction set computer (RISC) instruction set architecture (ISA)

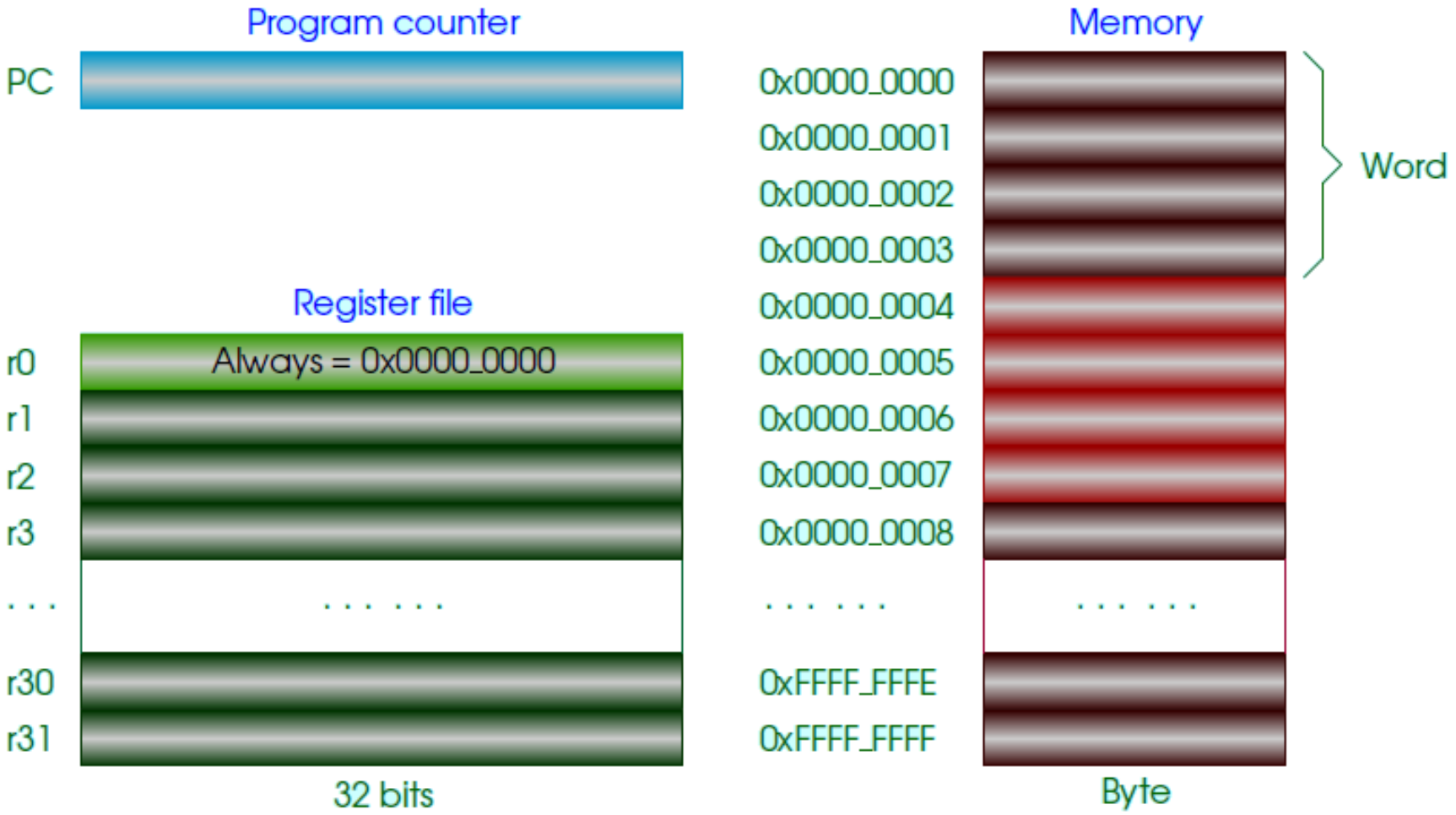
Pipeline


WPCIR
BRANCH
Control Unit
MM2REG
MWREG
MDESR
EDESR
EWREG
WREG
M2REG
WMEM
ALUC[3..0]
SHIFT
ALUIMM
RSRTEQU
SEXT
REGRT
FWDB[1..0]
FWDA[1..0]
JUMP
EWREG
EM2REG
EWMEM
EALUC[3..0]
EALUIMM
ESHIFT
MWREG
MM2REG
MWMEM
WWREG
WM2REG
func
op
rs
rt
address
imm
rs
rt
imm
rd
rt
4
PC
A
D
mem
N1
N2
RegFile
N
D
CK
Q1
Q2
WE
sa
SE
A
ALU
B
R
WE
A
mem
DI
DO
CLOCK
IF
ID
EXE
MEM
WB

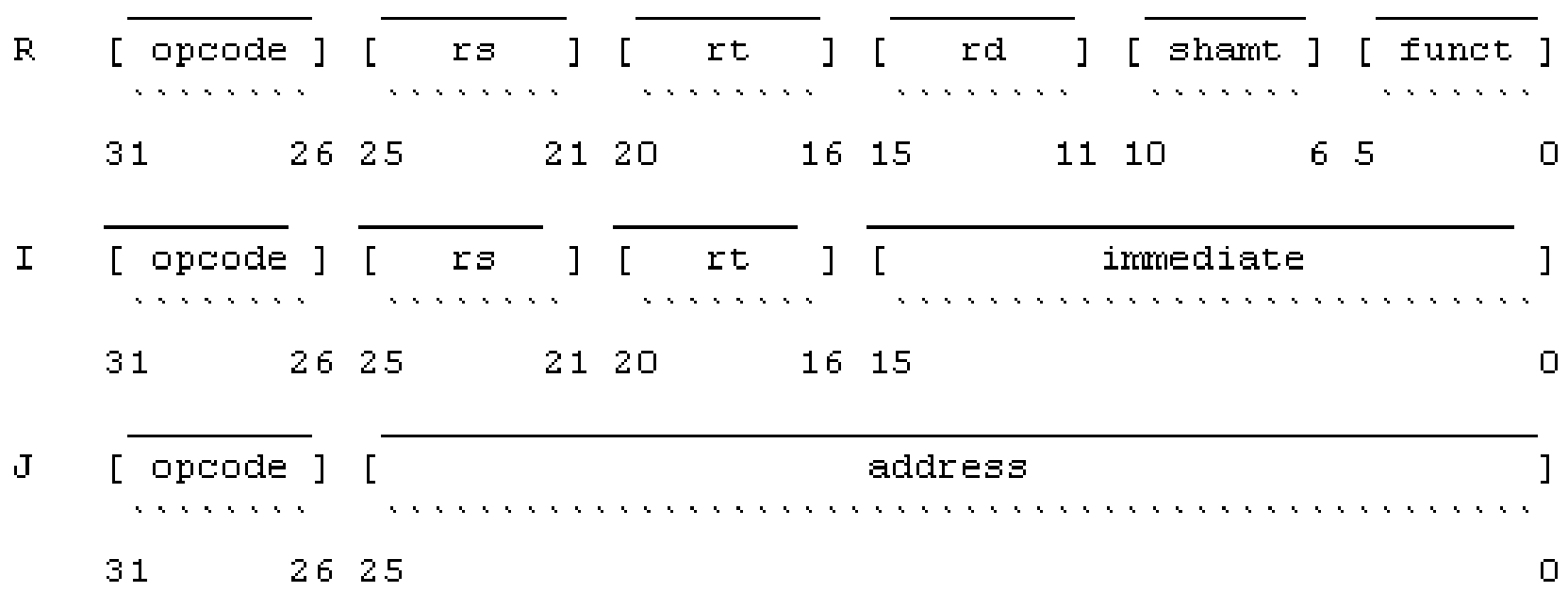
Main components



Registers && Memory

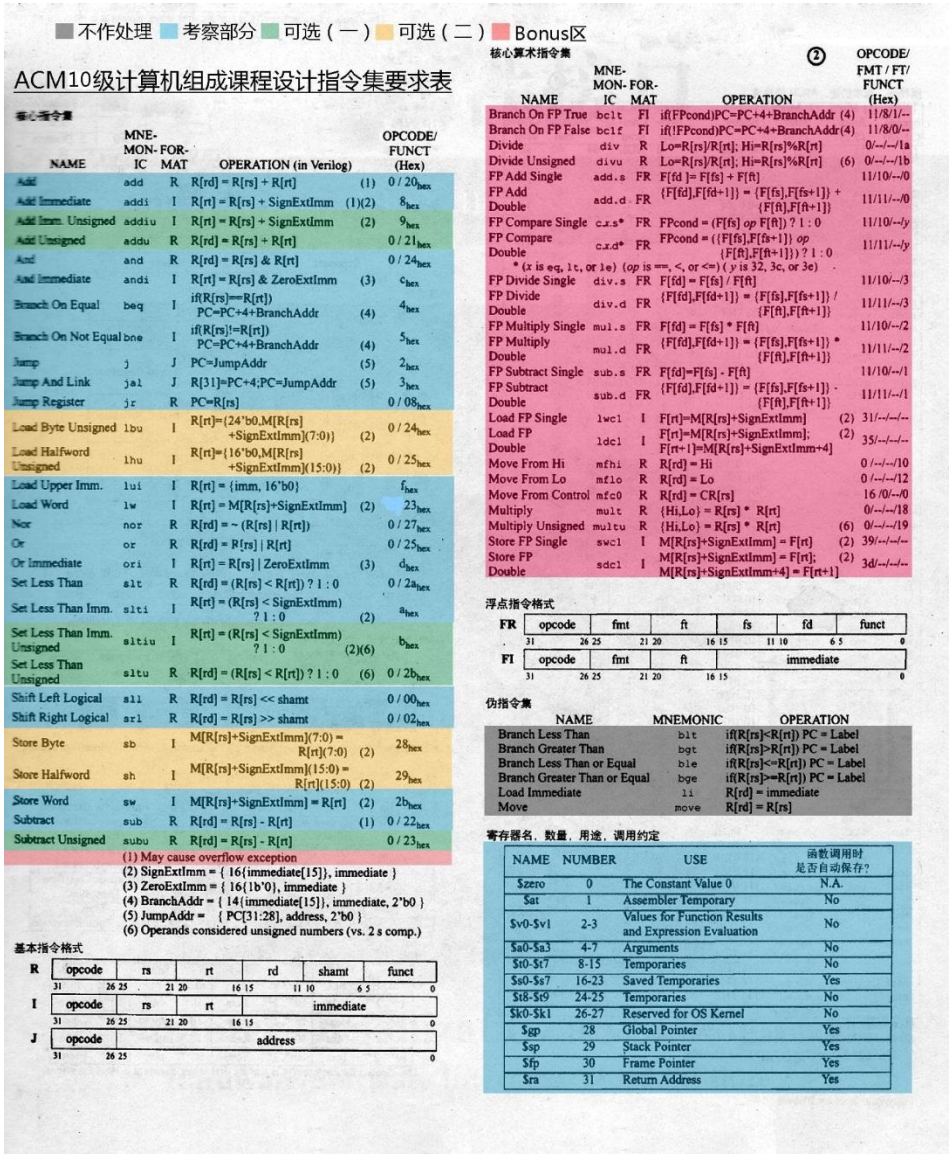


Instructions



Instructions(cont.)

- See Green Card



不作处理 考察部分 可选（一） 可选（二） Bonus区

ACM10级计算机组成课程设计指令集要求表

核心指令集

NAME	MNEMONIC	FORMAT	OPERATION (in Verilog)		OPCODE/FUNCT (Hex)
Add	add	R	R[rd] = R[rs] + R[rt]	(1)	0 / 20_{hex}
Add Immediate	addi	I	R[rt] = R[rs] + SignExtImm	(1)(2)	8_{hex}
Add Imm. Unsigned	addiu	I	R[rt] = R[rs] + SignExtImm	(2)	9_{hex}
Add Unsigned	addu	R	R[rd] = R[rs] + R[rt]		0 / 21_{hex}
And	and	R	R[rd] = R[rs] & R[rt]		0 / 24_{hex}
And Immediate	andi	I	R[rt] = R[rs] & ZeroExtImm	(3)	c_{hex}
Branch On Equal	beq	I	if(R[rs]==R[rt]) PC=PC+4+BranchAddr	(4)	4_{hex}
Branch On Not Equal	bne	I	if(R[rs]!=R[rt]) PC=PC+4+BranchAddr	(4)	5_{hex}
Jump	j	J	PC=JumpAddr	(5)	2_{hex}
Jump And Link	jal	J	R[31]=PC+4;PC=JumpAddr	(5)	3_{hex}
Jump Register	jr	R	PC=R[rs]		0 / 08_{hex}
Load Byte Unsigned	lbu	I	R[rt]={24'b0,M[R[rs]+SignExtImm](7:0)}	(2)	0 / 24_{hex}
Load Halfword Unsigned	lhu	I	R[rt]={16'b0,M[R[rs]+SignExtImm](15:0)}	(2)	0 / 25_{hex}
Load Upper Imm.	lui	I	R[rt] = {imm, 16'b0}		f_{hex}
Load Word	lw	I	R[rt] = M[R[rs]+SignExtImm]	(2)	23_{hex}
Nor	nor	R	R[rd] = ~ (R[rs] | R[rt])		0 / 27_{hex}
Or	or	R	R[rd] = R[rs] | R[rt]		0 / 25_{hex}
Or Immediate	ori	I	R[rt] = R[rs] | ZeroExtImm	(3)	d_{hex}
Set Less Than	slt	R	R[rd] = (R[rs] < R[rt]) ? 1 : 0		0 / $2a_{hex}$
Set Less Than Imm.	slti	I	R[rt] = (R[rs] < SignExtImm) ? 1 : 0	(2)	a_{hex}
Set Less Than Imm. Unsigned	sltiu	I	R[rt] = (R[rs] < SignExtImm) ? 1 : 0	(2)(6)	b_{hex}
Set Less Than Unsigned	sltu	R	R[rd] = (R[rs] < R[rt]) ? 1 : 0	(6)	0 / $2b_{hex}$
Shift Left Logical	sll	R	R[rd] = R[rs] << shamt		0 / 00_{hex}
Shift Right Logical	srl	R	R[rd] = R[rs] >> shamt		0 / 02_{hex}
Store Byte	sb	I	M[R[rs]+SignExtImm](7:0) = R[rt](7:0)	(2)	28_{hex}
Store Halfword	sh	I	M[R[rs]+SignExtImm](15:0) = R[rt](15:0)	(2)	29_{hex}
Store Word	sw	I	M[R[rs]+SignExtImm] = R[rt]	(2)	$2b_{hex}$
Subtract	sub	R	R[rd] = R[rs] - R[rt]	(1)	0 / 22_{hex}
Subtract Unsigned	subu	R	R[rd] = R[rs] - R[rt]		0 / 23_{hex}

(1) May cause overflow exception
(2) SignExtImm = { 16{immediate[15]}, immediate }
(3) ZeroExtImm = { 16{1'b0}, immediate }
(4) BranchAddr = { 14{immediate[15]}, immediate, 2'b0 }
(5) JumpAddr = { PC[31:28], address, 2'b0 }
(6) Operands considered unsigned numbers (vs. 2 s comp.)

基本指令格式

核心算术指令集

NAME	MNEMONIC	FORMAT	OPERATION		OPCODE/FMT / FT/ FUNCT (Hex)
Branch On FP True	bc1t	FI	if(FPcond)PC=PC+4+BranchAddr	(4)	11/8/1/--
Branch On FP False	bc1f	FI	if(!FPcond)PC=PC+4+BranchAddr	(4)	11/8/0/--
Divide	div	R	Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt]		0/--/--/1a
Divide Unsigned	divu	R	Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt]	(6)	0/--/--/1b
FP Add Single	add.s	FR	F[fd]= F[fs] + F[ft]		11/10/--/0
FP Add Double	add.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} + {F[ft],F[ft+1]}		11/11/--/0
FP Compare Single	c.x.s*	FR	FPcond = (F[fs] *op* F[ft]) ? 1 : 0		11/10/--/*y*
FP Compare Double	c.x.d*	FR	FPcond = ({F[fs],F[fs+1]} *op* {F[ft],F[ft+1]}) ? 1 : 0		11/11/--/*y*
* (*x* is eq, lt, or le) (*op* is ==, <, or <=) (*y* is 32, 3c, or 3e)					
FP Divide Single	div.s	FR	F[fd] = F[fs] / F[ft]		11/10/--/3
FP Divide Double	div.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} / {F[ft],F[ft+1]}		11/11/--/3
FP Multiply Single	mul.s	FR	F[fd] = F[fs] * F[ft]		11/10/--/2
FP Multiply Double	mul.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} * {F[ft],F[ft+1]}		11/11/--/2
FP Subtract Single	sub.s	FR	F[fd]=F[fs] - F[ft]		11/10/--/1
FP Subtract Double	sub.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} - {F[ft],F[ft+1]}		11/11/--/1
Load FP Single	lwc1	I	F[rt]=M[R[rs]+SignExtImm]	(2)	31/--/--/--
Load FP Double	ldc1	I	F[rt]=M[R[rs]+SignExtImm]; F[rt+1]=M[R[rs]+SignExtImm+4]	(2)	35/--/--/--
Move From Hi	mfhi	R	R[rd] = Hi		0 /--/--/10
Move From Lo	mflo	R	R[rd] = Lo		0 /--/--/12
Move From Control	mfc0	R	R[rd] = CR[rs]		16 /0/--/0
Multiply	mult	R	{Hi,Lo} = R[rs] * R[rt]		0/--/--/18
Multiply Unsigned	multu	R	{Hi,Lo} = R[rs] * R[rt]	(6)	0/--/--/19
Store FP Single	swc1	I	M[R[rs]+SignExtImm] = F[rt]	(2)	39/--/--/--
Store FP Double	sdc1	I	M[R[rs]+SignExtImm] = F[rt]; M[R[rs]+SignExtImm+4] = F[rt+1]	(2)	3d/--/--/--

浮点指令格式

伪指令集

NAME	MNEMONIC	OPERATION
Branch Less Than	blt	if(R[rs]<R[rt]) PC = Label
Branch Greater Than	bgt	if(R[rs]>R[rt]) PC = Label
Branch Less Than or Equal	ble	if(R[rs]<=R[rt]) PC = Label
Branch Greater Than or Equal	bge	if(R[rs]>=R[rt]) PC = Label
Load Immediate	li	R[rd] = immediate
Move	move	R[rd] = R[rs]

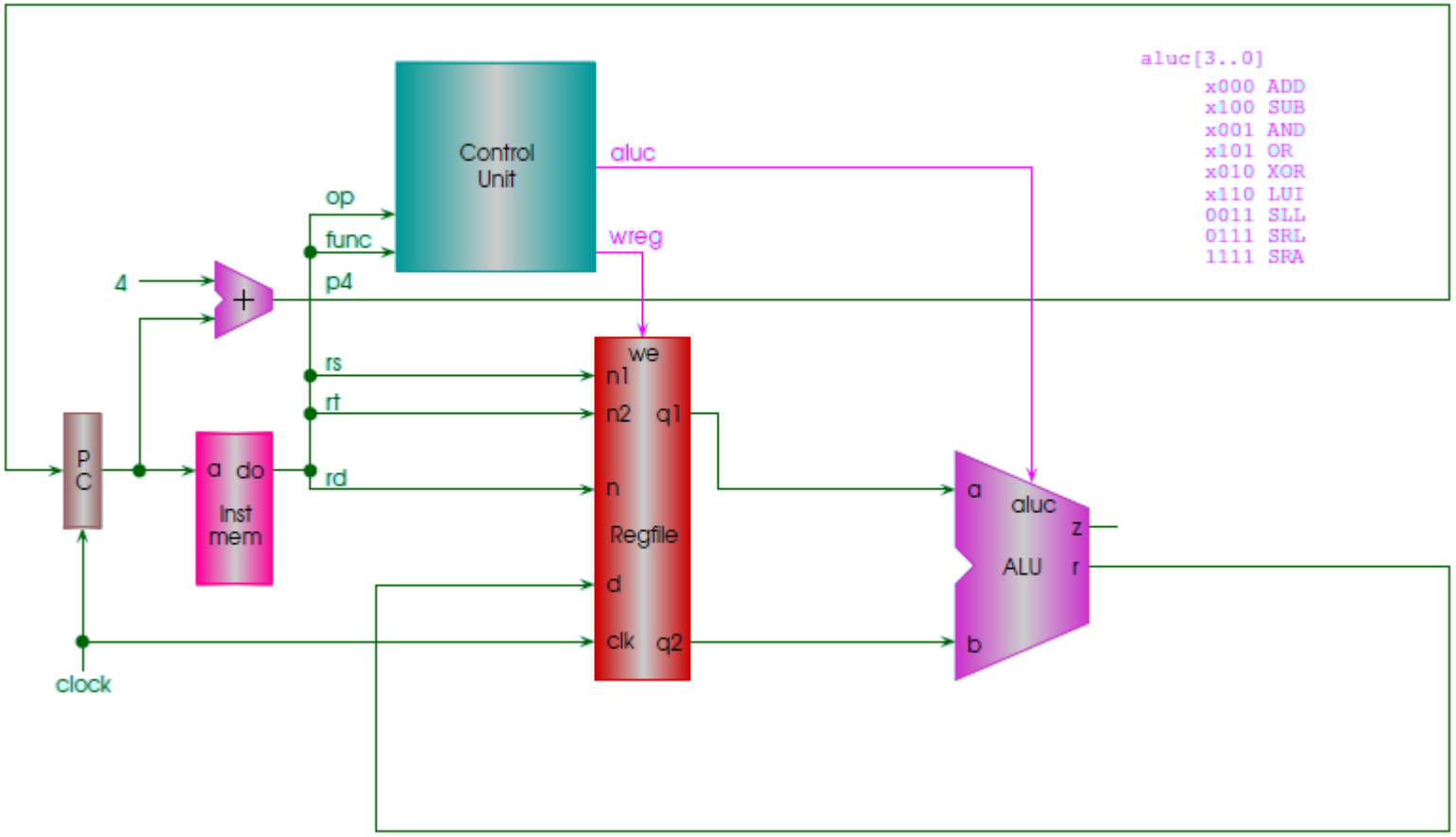
寄存器名，数量，用途，调用约定

NAME	NUMBER	USE	函数调用时是否自动保存?
$zero	0	The Constant Value 0	N.A.
$at	1	Assembler Temporary	No
$v0-$v1	2-3	Values for Function Results and Expression Evaluation	No
$a0-$a3	4-7	Arguments	No
$t0-$t7	8-15	Temporaries	No
$s0-$s7	16-23	Saved Temporaries	Yes
$t8-$t9	24-25	Temporaries	No
$k0-$k1	26-27	Reserved for OS Kernel	No
$gp	28	Global Pointer	Yes
$sp	29	Stack Pointer	Yes
$fp	30	Frame Pointer	Yes
$ra	31	Return Address	Yes

Instructions(example)

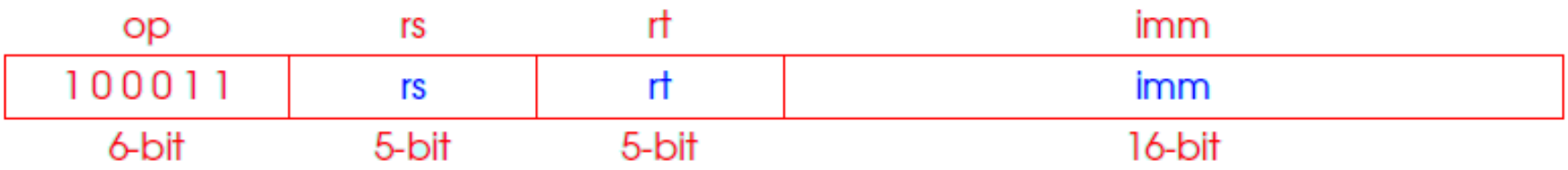
```
add  rd, rs, rt  ; rd = rs + rt
```

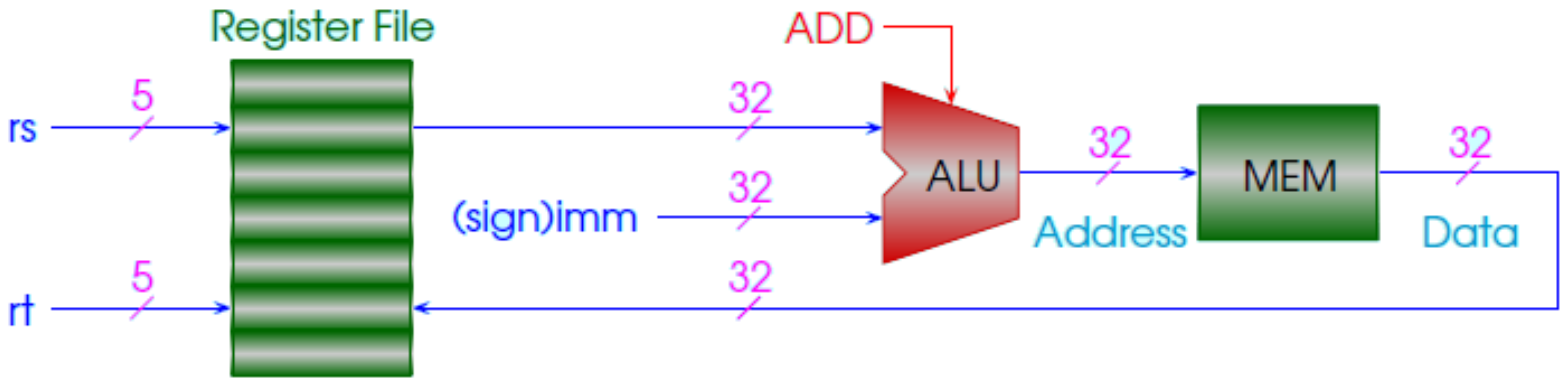
op	rs	rt	rd	sa	funct
000000	rs	rt	rd	00000	100000
6-bit	5-bit	5-bit	5-bit	5-bit	6-bit



Instructions(example)

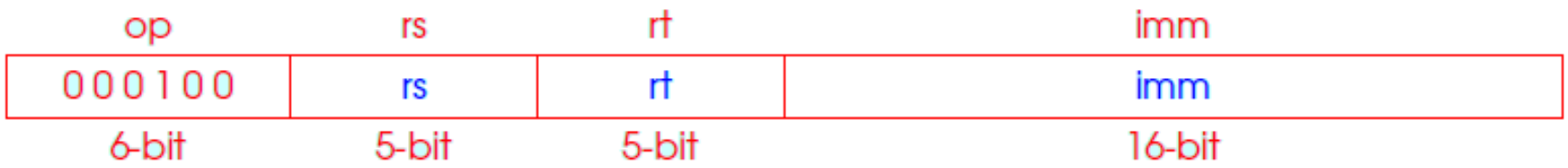
```
lw   rt, imm(rs)   ; rt <-- memory[rs + (sign)imm]
```

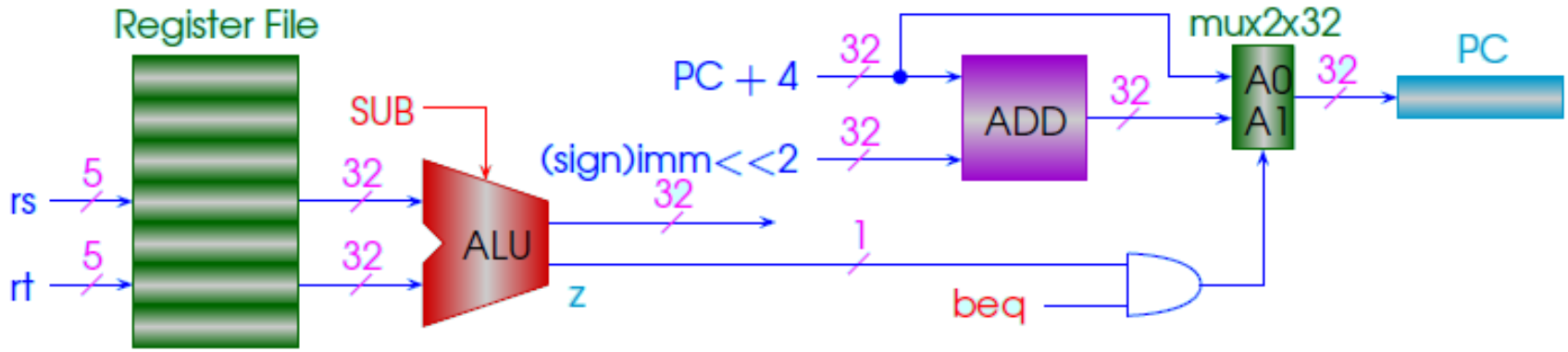




Instructions(example)

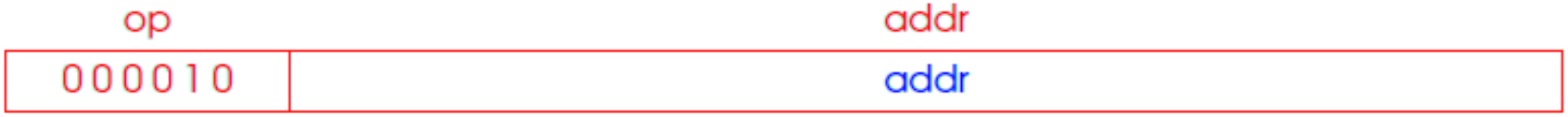
```
beq rs, rt, imm; if(rs==rt)PC<--PC+4+(sign)imm<<2
```

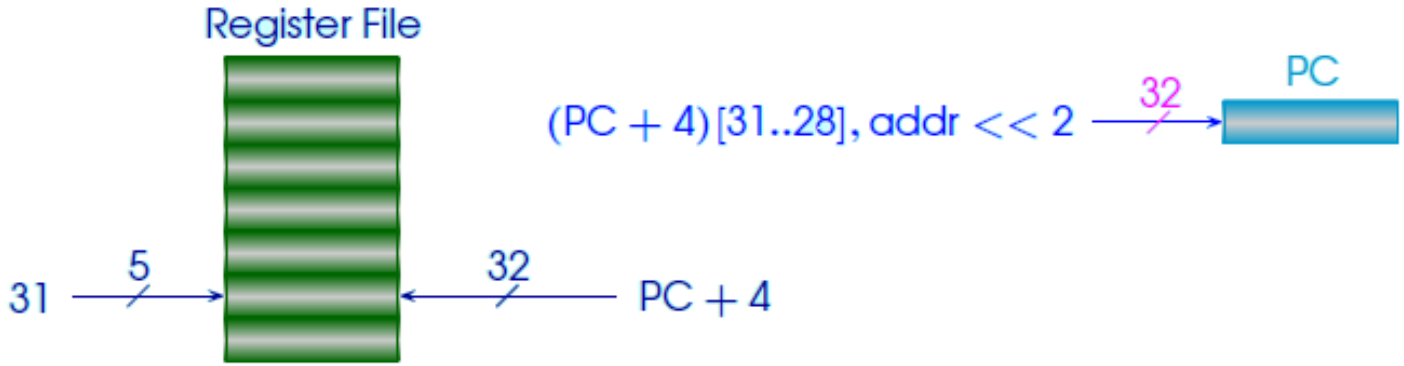




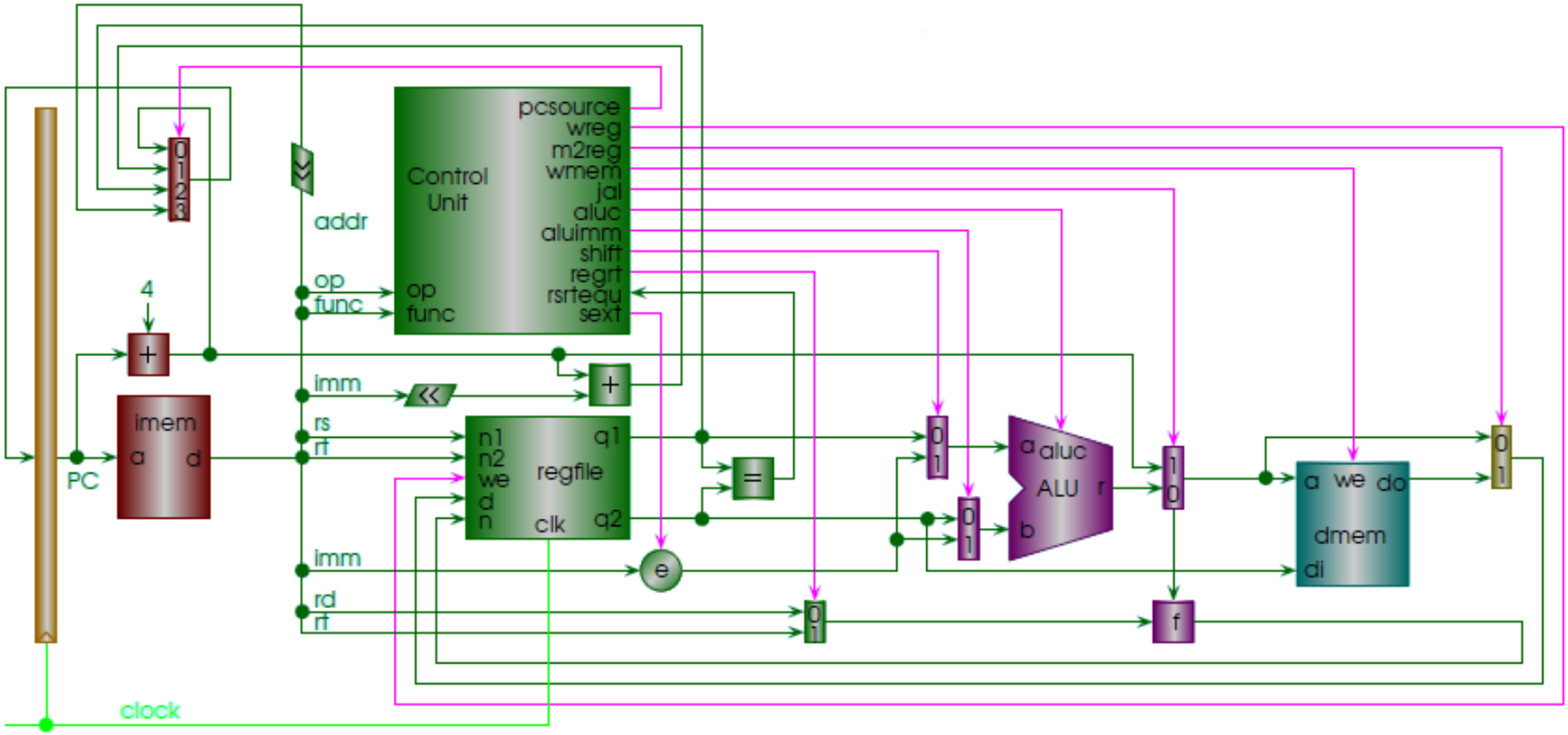
Instructions(example)

```
j    addr          ; PC<--(PC+4)[31..28],addr<<2
```

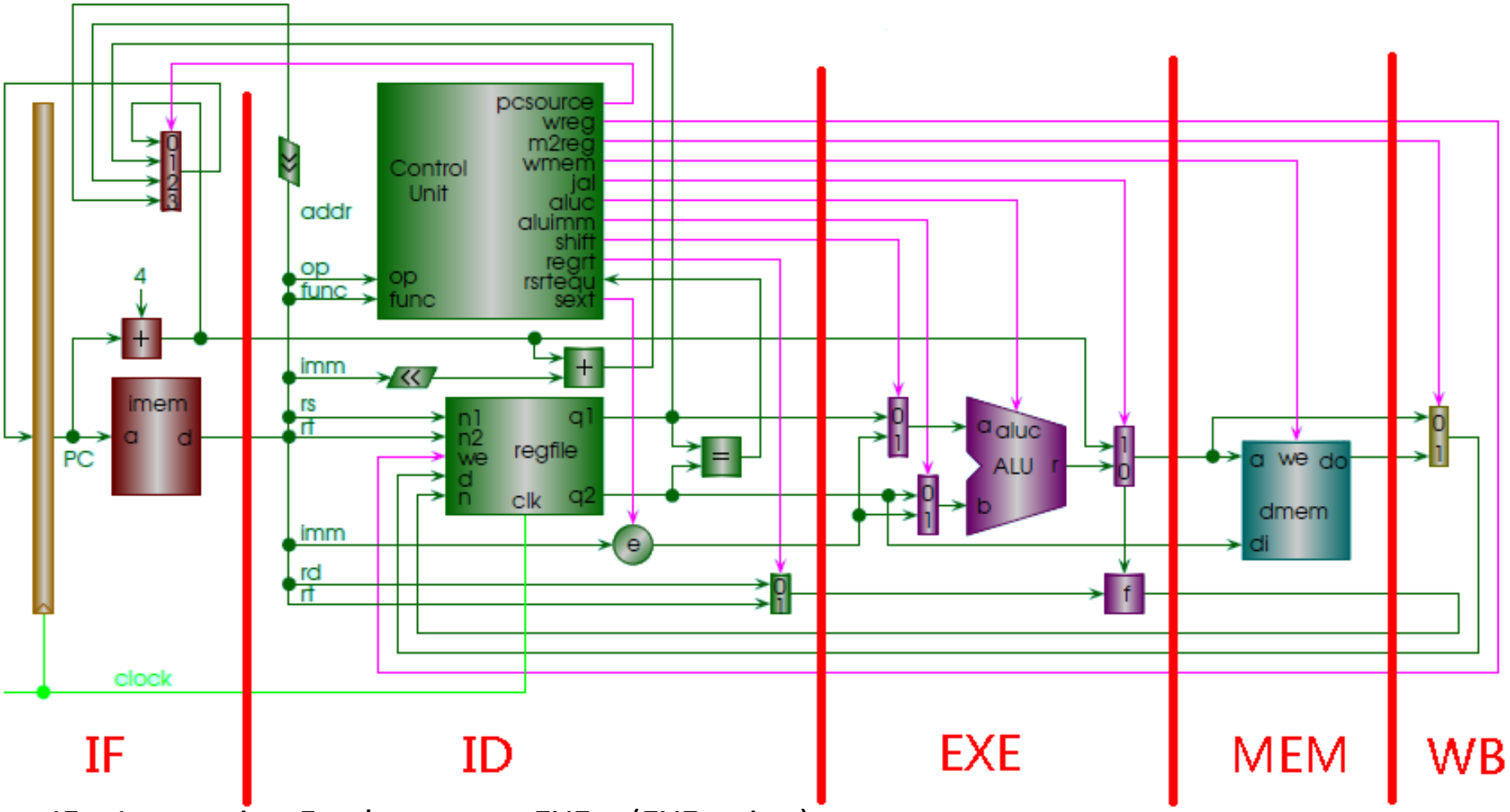




Single sycle CPU



Cut them into stages



IF – Instruction Fetch

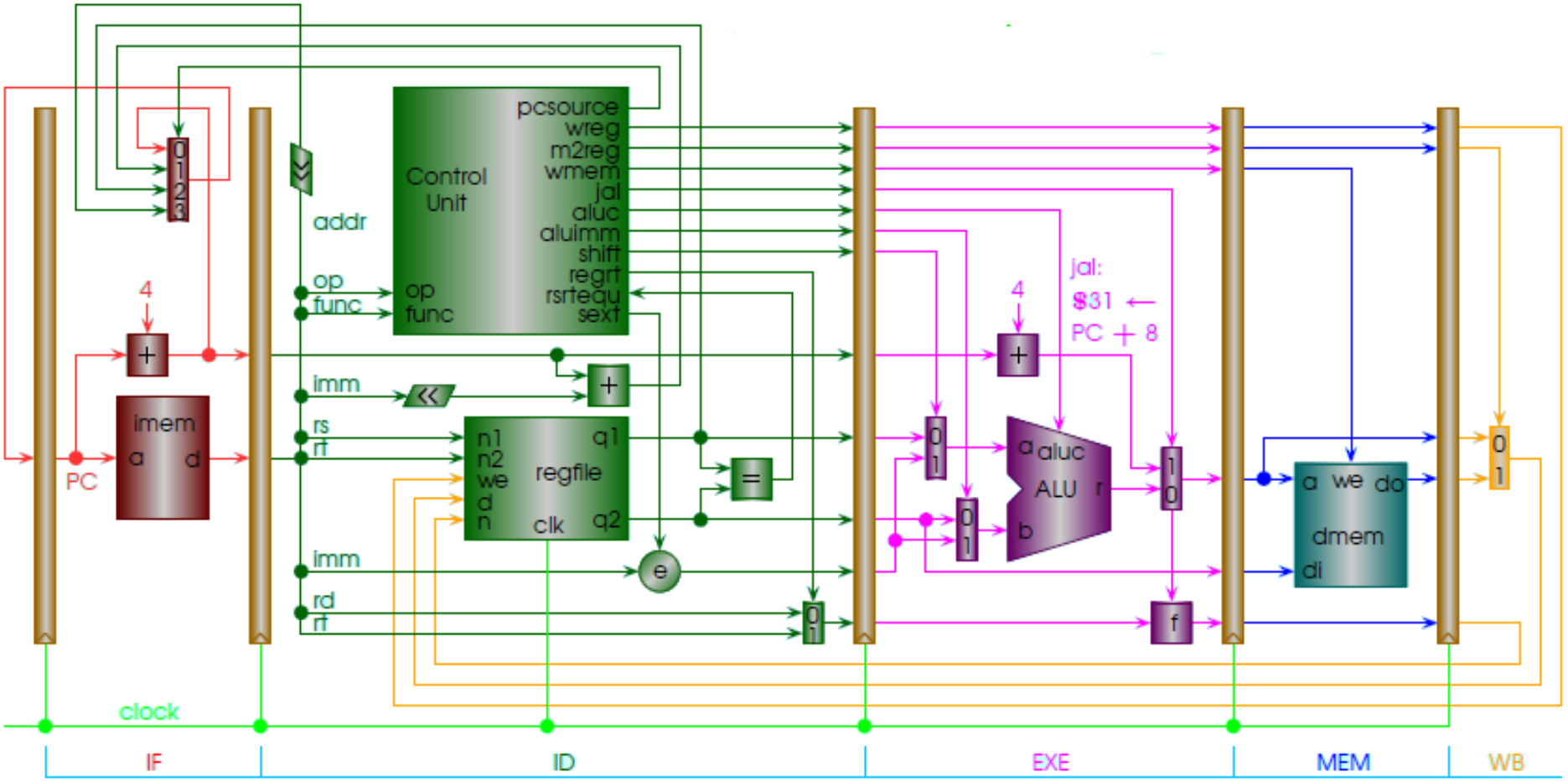
ID – Instruction Decode

EXE – (EXEcution)

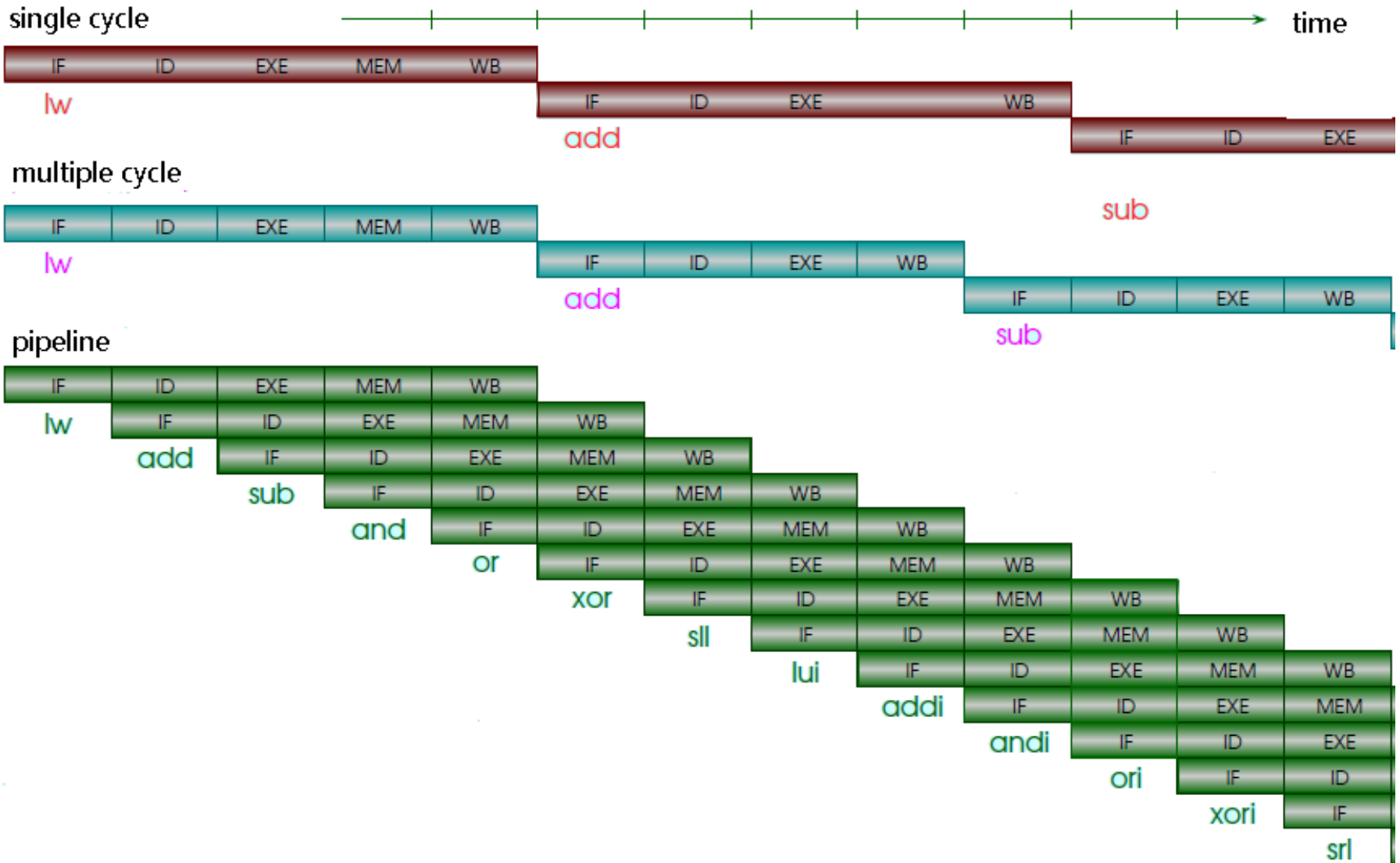
MEM – (MEMory access)

WB – (Write Back)

baseline


pcsource
wreg
m2reg
wmem
jal
aluc
aluimm
shift
regrt
rsrtequ
sext
Control Unit
op
func
addr
imm
rs
rt
rd
imm
4
imem
a
d
PC
n1
n2
we
d
n
clk
q1
q2
regfile
=
e
jal:
$31 ← PC + 8
a
aluc
ALU
r
b
we
do
dmem
di
clock
IF
ID
EXE
MEM
WB

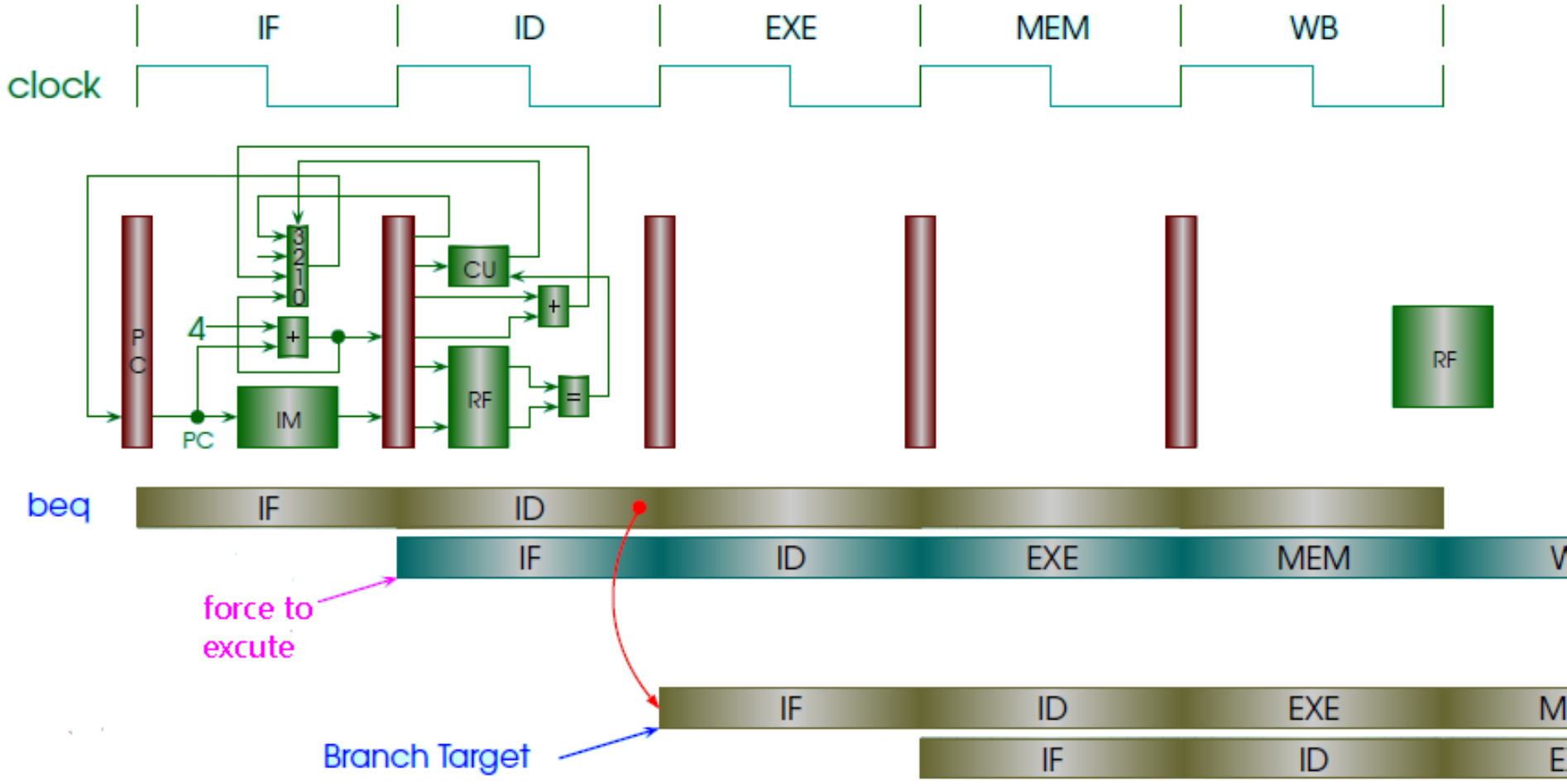
How pipeline works



hazard

- Structural dependency
  - Add hardwares
  - Compiler
- Control dependency
  - Delayed branch
  - *Dynamic branch prediction
- Data dependency
  - Compiler
  - Forwarding
  - Stall

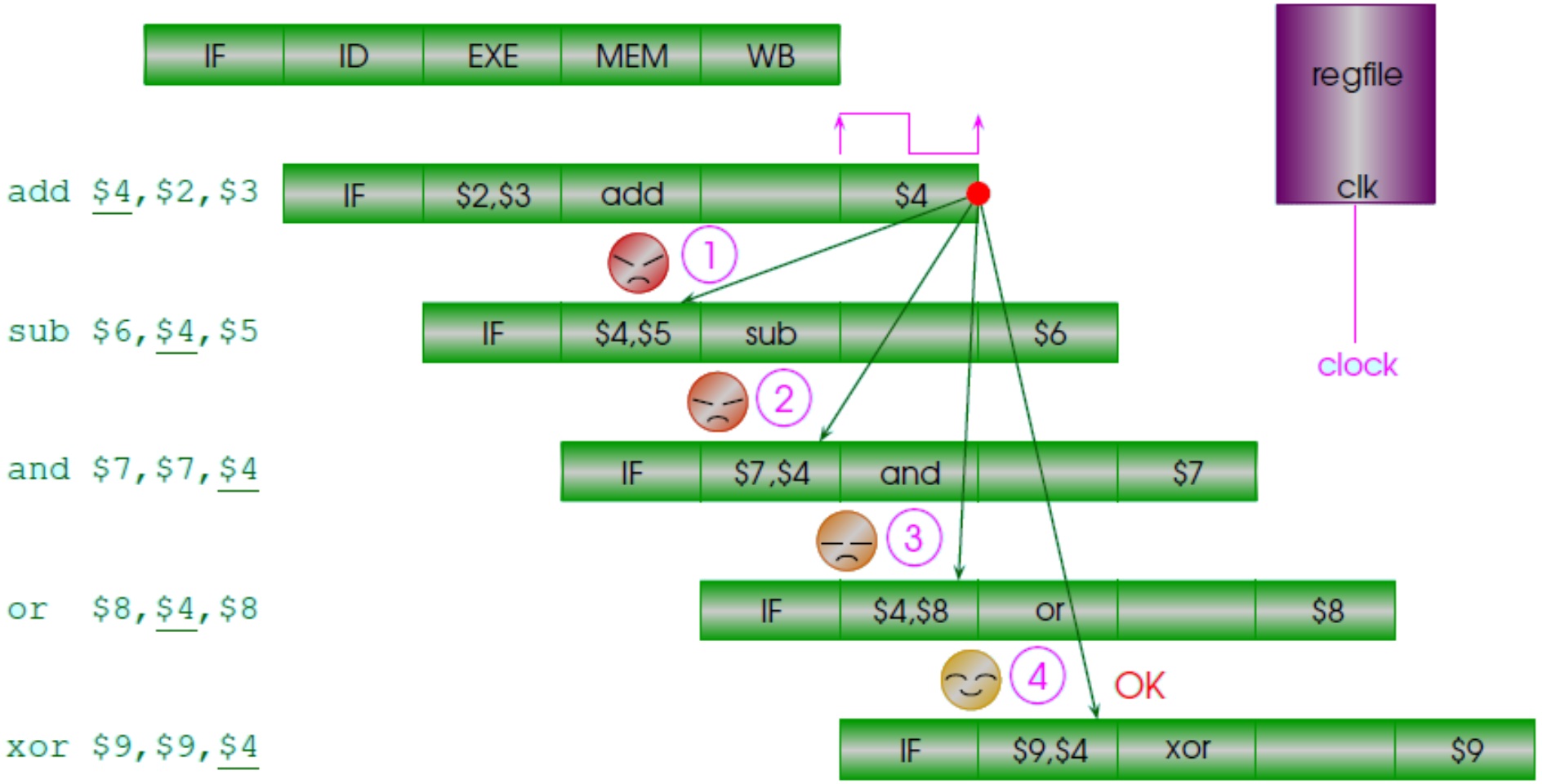
Delayed branch



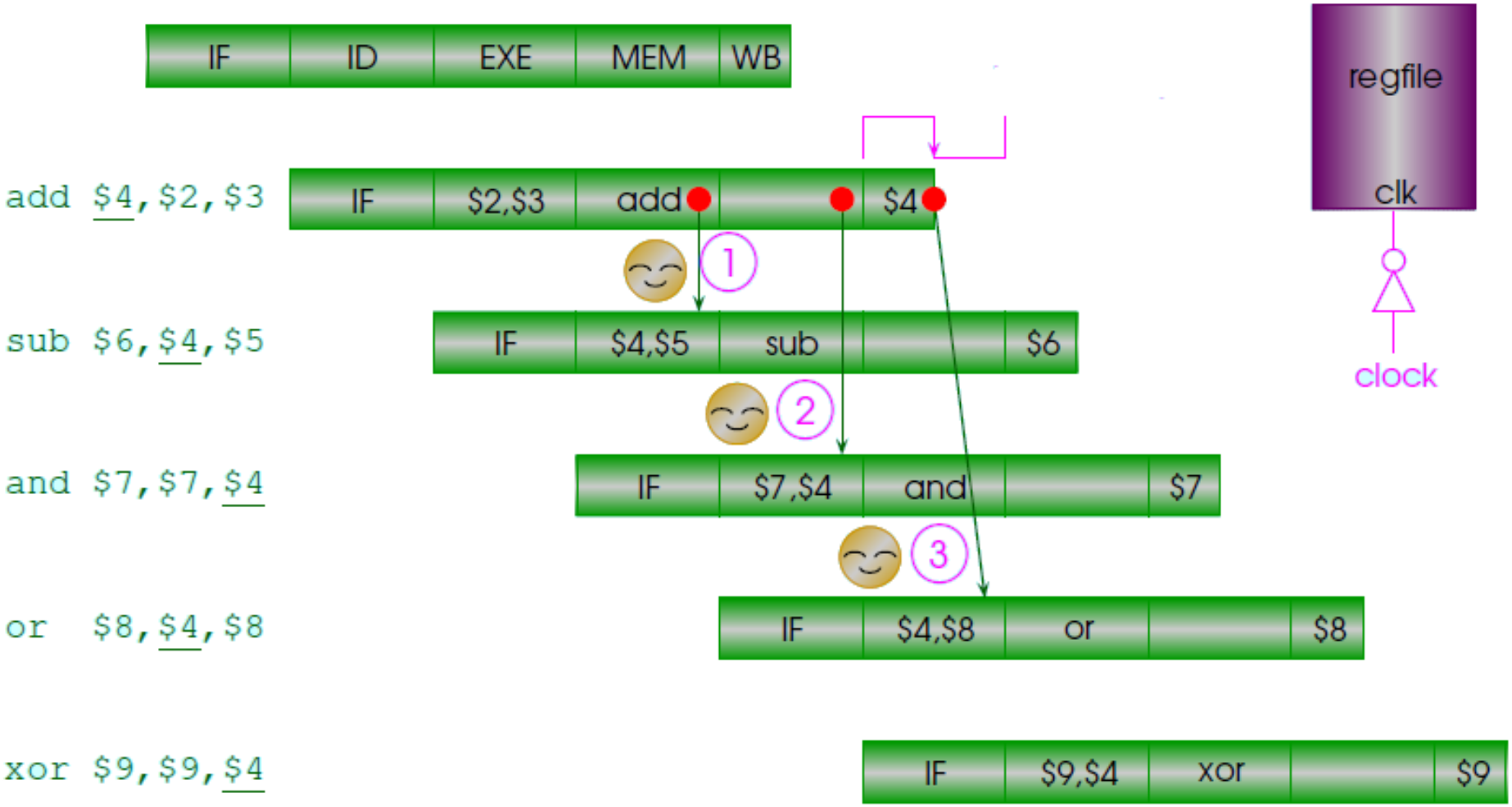
Ambiguity

- In jal, the return address in $ra is PC+4 instead of PC+8

Data dependency

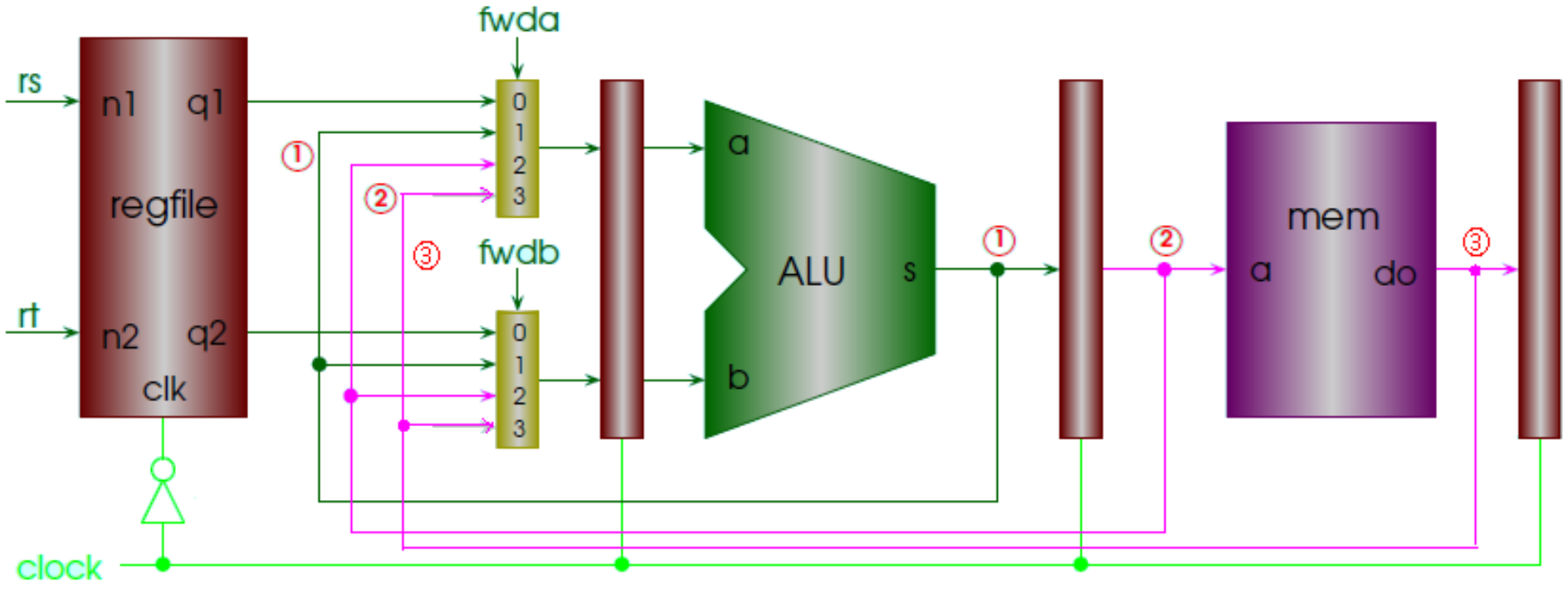


Forwarding

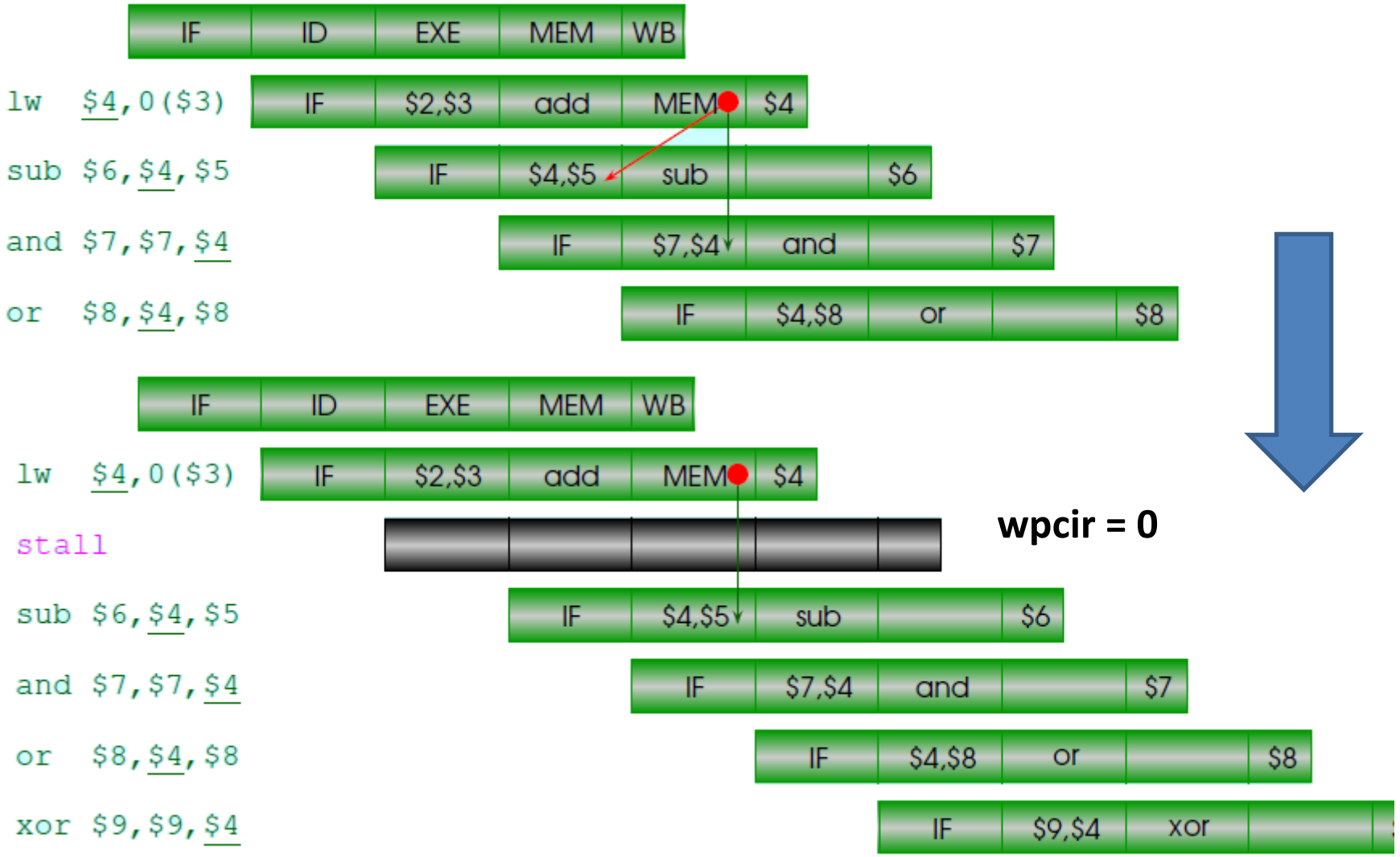

IF
ID
EXE
MEM
WB
regfile
clk
clock
add $4,$2,$3
IF
$2,$3
add
$4
1
sub $6,$4,$5
IF
$4,$5
sub
$6
2
and $7,$7,$4
IF
$7,$4
and
$7
3
or $8,$4,$8
IF
$4,$8
or
$8
xor $9,$9,$4
IF
$9,$4
xor
$9

3 kinds of forwardings

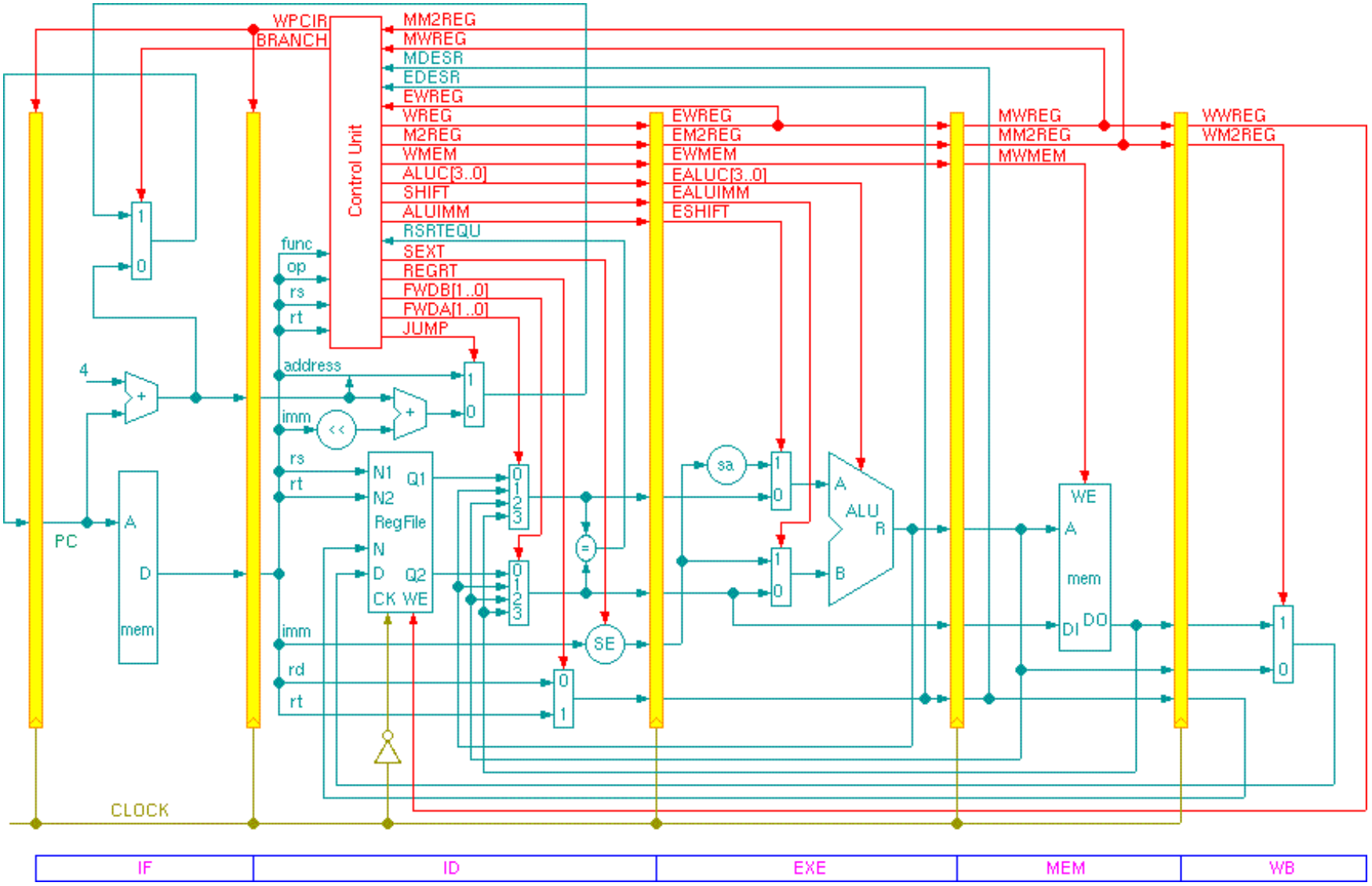
- Forward ALU result from EXE stage to ID stage
- Forward ALU result from MEM stage to ID stage
- Forward MEM result from MEM stage to ID stage



Stall


IF ID EXE MEM WB
lw $4,0($3)
IF $2,$3 add MEM $4
sub $6,$4,$5
IF $4,$5 sub $6
and $7,$7,$4
IF $7,$4 and $7
or $8,$4,$8
IF $4,$8 or $8
IF ID EXE MEM WB
lw $4,0($3)
IF $2,$3 add MEM $4
stall
wpcir = 0
sub $6,$4,$5
IF $4,$5 sub $6
and $7,$7,$4
IF $7,$4 and $7
or $8,$4,$8
IF $4,$8 or $8
xor $9,$9,$4
IF $9,$4 xor

Pipeline


WPCIR
BRANCH
Control Unit
MM2REG
MWREG
MDESR
EDESR
EWREG
WREG
M2REG
WMEM
ALUC[3..0]
SHIFT
ALUIMM
RSRTEQU
SEXT
REGRT
FWDB[1..0]
FWDA[1..0]
JUMP
EWREG
EM2REG
EWMEM
EALUC[3..0]
EALUIMM
ESHIFT
MWREG
MM2REG
MWMEM
WWREG
WM2REG
func
op
rs
rt
address
imm
rs
rt
imm
rd
rt
4
PC
A
D
mem
N1
N2
RegFile
N
D
CK
Q1
Q2
WE
sa
SE
A
ALU
B
R
WE
A
mem
DI
DO
CLOCK
IF
ID
EXE
MEM
WB

Parameters

- Cycle： 4(2*2) clocks / cycle
  - ‾‾ |__| ‾‾ |__
  - always #2 clock = ~clock;（bonus: #1）

- Memory: 10 clocks but **no delay in this phase**
  - instructions and data share the same memory
- 1st level cache: 3 clocks
- 2nd level cache: 5 clocks

Self-modifying code

- code that alters its own instructions while it is executing

- See wikipedia for more information
  - http://en.wikipedia.org/wiki/Self-modifying_code

A simple demo

How to start

- http://mips.acm-project.org/
- Learn Verilog.
- Install Modelsim.
- Slides of liyamin

Some other words…

- This is an open project, try to shock TAs.
- NO CHEAT.
- 出来混，总是要还的。
- 安迪爬过了一条又臭又长的排水管，干干净净地到另一个世界去了。
- Enjoy it.

Q & A