MIPS Computer

COMP 411
Apr 21, 2017

Istvan Csapo

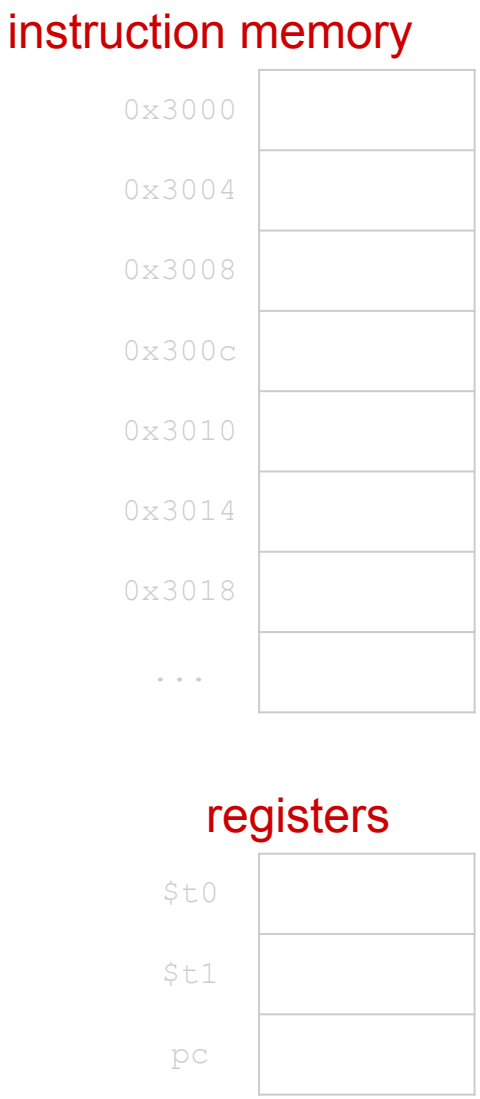
Simple MIPS Program

C code

```c
int main() {
  int x = 1;
  int y = x + 1;
}
```

MIPS code

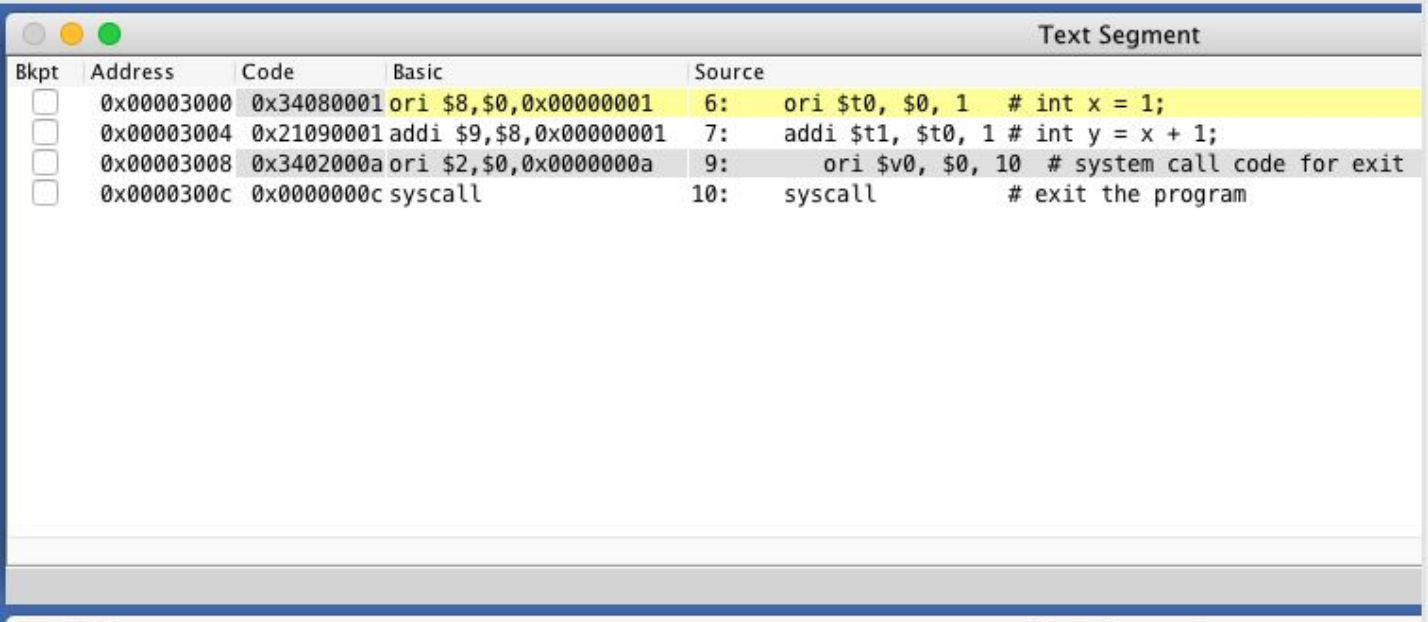
```
.data 0x0

.text 0x3000
main:
  ori  $t0, $0, 1  # int x = 1;
  addi $t1, $t0, 1 # int y = x + 1;

  ori $v0, $0, 10  # system call code for exit
  syscall          # exit the program
```



Program in MARS

What happens in MARS?

```
1  .data 0x0
2
3  .text 0x3000
4  main:
5    ori  $t0, $0, 1  # int x = 1;
6    addi $t1, $t0, 1 # int y = x + 1;
7
8    ori $v0, $0, 10  # system call code for exit
9    syscall          # exit the program
```



Text Segment

Bkpt	Address	Code	Basic	Source
☐	0x00003000	0x34080001	ori $8,$0,0x00000001	6: ori $t0, $0, 1 # int x = 1;
☐	0x00003004	0x21090001	addi $9,$8,0x00000001	7: addi $t1, $t0, 1 # int y = x + 1;
☐	0x00003008	0x3402000a	ori $2,$0,0x0000000a	9: ori $v0, $0, 10 # system call code for exit
☐	0x0000300c	0x0000000c	syscall	10: syscall # exit the program

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)
0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x00000020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x00000040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x00000060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x00000080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x000000a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x000000c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x000000e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x00000100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x00000120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x00000140	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x00000160	0x00000000	0x00000000	0x00000000	0x00000000	0x00000

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000000
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000000
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x00000000
$s1	17	0x00000000
$s2	18	0x00000000
$s3	19	0x00000000
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x00001800
$sp	29	0x00002ffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x00003000
hi		0x00000000
lo		0x00000000

Program in MARS

What happens in MARS?

```
1  .data 0x0
2
3  .text 0x3000
4  main:
5    ori  $t0, $0, 1  # int x = 1;
6    addi $t1, $t0, 1 # int y = x + 1;
7
8    ori $v0, $0, 10  # system call code for exit
9    syscall          # exit the program
```

Text Segment

Bkpt	Address	Code	Basic	Source
☐	0x00003000	0x34080001	ori $8,$0,0x00000001	6: ori $t0, $0, 1 # int x = 1;
☐	0x00003004	0x21090001	addi $9,$8,0x00000001	7: addi $t1, $t0, 1 # int y = x + 1;
☐	0x00003008	0x3402000a	ori $2,$0,0x0000000a	9: ori $v0, $0, 10 # system call code for exit
☐	0x0000300c	0x0000000c	syscall	10: syscall # exit the program

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)
0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x00000020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x00000040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x00000060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x00000080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x000000a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x000000c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x000000e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x00000100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x00000120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x00000140	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x00000160	0x00000000	0x00000000	0x00000000	0x00000000	0x00000

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000000
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000001
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x00000000
$s1	17	0x00000000
$s2	18	0x00000000
$s3	19	0x00000000
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x00001800
$sp	29	0x00002ffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x00003004
hi		0x00000000
lo		0x00000000

Program in MARS

What happens in MARS?

```
1  .data 0x0
2
3  .text 0x3000
4  main:
5    ori  $t0, $0, 1  # int x = 1;
6    addi $t1, $t0, 1 # int y = x + 1;
7
8    ori $v0, $0, 10  # system call code for exit
9    syscall          # exit the program
```

Text Segment

Bkpt	Address	Code	Basic	Source
☐	0x00003000	0x34080001	ori $8,$0,0x00000001	6: ori $t0, $0, 1 # int x = 1;
☐	0x00003004	0x21090001	addi $9,$8,0x00000001	7: addi $t1, $t0, 1 # int y = x + 1;
☐	0x00003008	0x3402000a	ori $2,$0,0x0000000a	9: ori $v0, $0, 10 # system call code for exit
☐	0x0000300c	0x0000000c	syscall	10: syscall # exit the program

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)
0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x00000020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x00000040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x00000060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x00000080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x000000a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x000000c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x000000e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x00000100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x00000120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x00000140	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x00000160	0x00000000	0x00000000	0x00000000	0x00000000	0x00000

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000000
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000001
$t1	9	0x00000002
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x00000000
$s1	17	0x00000000
$s2	18	0x00000000
$s3	19	0x00000000
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x00001800
$sp	29	0x00002ffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x00003008
hi		0x00000000
lo		0x00000000

Program in MARS

What happens in MARS?

```
1  .data 0x0
2
3  .text 0x3000
4  main:
5    ori  $t0, $0, 1  # int x = 1;
6    addi $t1, $t0, 1 # int y = x + 1;
7
8    ori $v0, $0, 10  # system call code for exit
9    syscall          # exit the program
```

Text Segment

Bkpt	Address	Code	Basic	Source
☐	0x00003000	0x34080001	ori $8,$0,0x00000001	6: ori $t0, $0, 1 # int x = 1;
☐	0x00003004	0x21090001	addi $9,$8,0x00000001	7: addi $t1, $t0, 1 # int y = x + 1;
☐	0x00003008	0x3402000a	ori $2,$0,0x0000000a	9: ori $v0, $0, 10 # system call code for exit
☐	0x0000300c	0x0000000c	syscall	10: syscall # exit the program

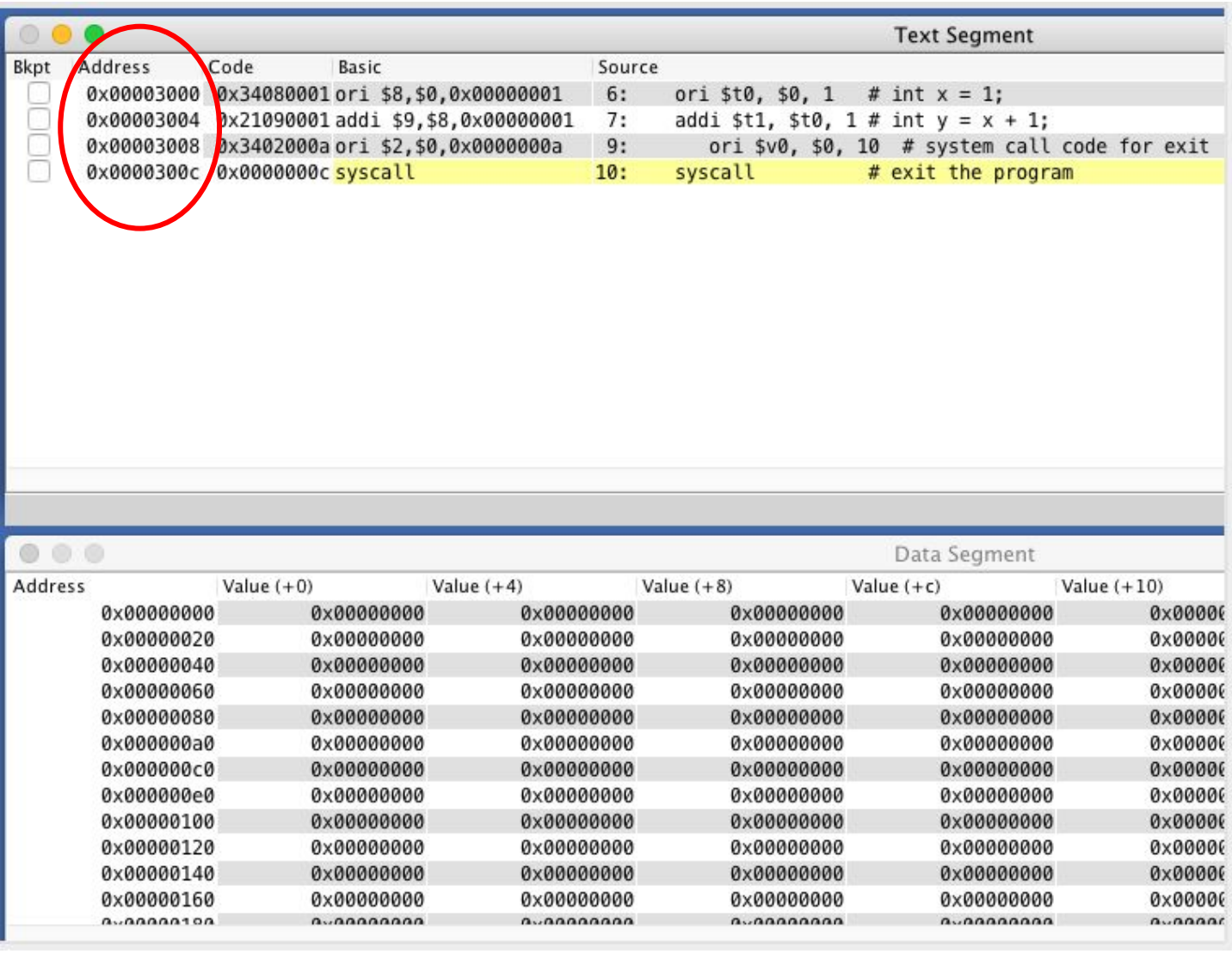
Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)
0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x00000020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x00000040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x00000060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x00000080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x000000a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x000000c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x000000e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x00000100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x00000120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x00000140	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x00000160	0x00000000	0x00000000	0x00000000	0x00000000	0x00000

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x0000000a
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000001
$t1	9	0x00000002
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x00000000
$s1	17	0x00000000
$s2	18	0x00000000
$s3	19	0x00000000
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x00001800
$sp	29	0x00002ffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x0000300c
hi		0x00000000
lo		0x00000000

Program in MARS

MARS shows the instruction memory address and the PC



Text Segment

Bkpt	Address	Code	Basic	Source
	0x00003000	0x34080001	ori $8,$0,0x00000001	6: ori $t0, $0, 1 # int x = 1;
	0x00003004	0x21090001	addi $9,$8,0x00000001	7: addi $t1, $t0, 1 # int y = x + 1;
	0x00003008	0x3402000a	ori $2,$0,0x0000000a	9: ori $v0, $0, 10 # system call code for exit
	0x0000300c	0x0000000c	syscall	10: syscall # exit the program

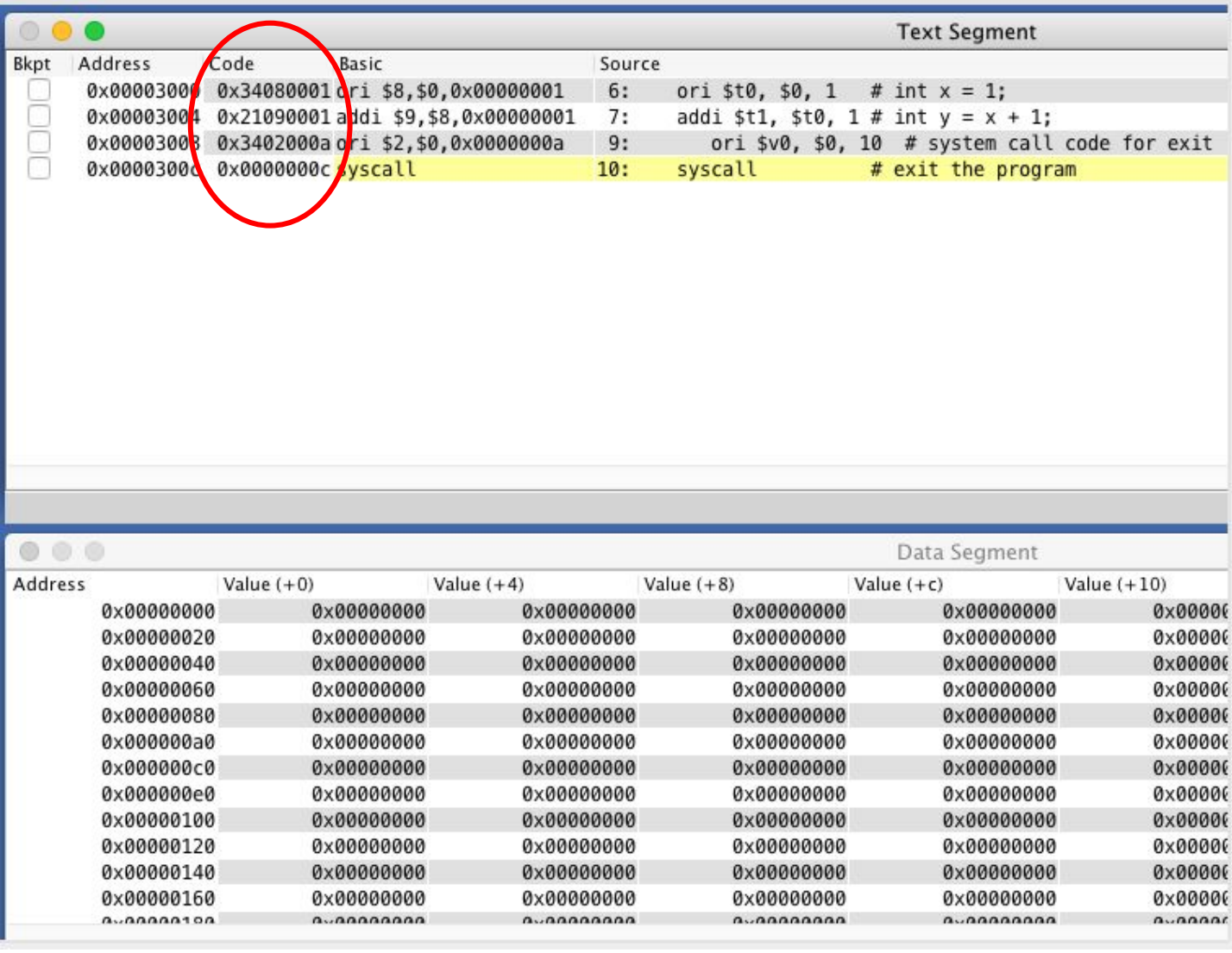
Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)
0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x00000020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x00000040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x00000060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x00000080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x000000a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x000000c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x000000e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x00000100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x00000120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x00000140	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x00000160	0x00000000	0x00000000	0x00000000	0x00000000	0x00000

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x0000000a
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000001
$t1	9	0x00000002
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x00000000
$s1	17	0x00000000
$s2	18	0x00000000
$s3	19	0x00000000
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x00001800
$sp	29	0x00002ffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x0000300c
hi		0x00000000
lo		0x00000000

Program in MARS

MARS also shows the OP codes for each instruction



Text Segment

Bkpt	Address	Code	Basic	Source
	0x00003000	0x34080001	ori $8,$0,0x00000001	6: ori $t0, $0, 1 # int x = 1;
	0x00003004	0x21090001	addi $9,$8,0x00000001	7: addi $t1, $t0, 1 # int y = x + 1;
	0x00003008	0x3402000a	ori $2,$0,0x0000000a	9: ori $v0, $0, 10 # system call code for exit
	0x0000300c	0x0000000c	syscall	10: syscall # exit the program

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)
0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x00000020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x00000040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x00000060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x00000080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x000000a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x000000c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x000000e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x00000100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x00000120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x00000140	0x00000000	0x00000000	0x00000000	0x00000000	0x00000
0x00000160	0x00000000	0x00000000	0x00000000	0x00000000	0x00000

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x0000000a
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000001
$t1	9	0x00000002
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x00000000
$s1	17	0x00000000
$s2	18	0x00000000
$s3	19	0x00000000
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x00001800
$sp	29	0x00002ffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x0000300c
hi		0x00000000
lo		0x00000000

Simple MIPS Program

Our program in memory after compilation

C code

```
1  int main() {
2    int x = 1;
3    int y = x + 1;
4  }
```

MIPS code

```
1           .data 0x0
2
3           .text 0x3000
4           main:
5  0x3000     ori  $t0, $0, 1  # int x = 1;
6  0x3004     addi $t1, $t0, 1 # int y = x + 1;
7
8  0x3008     ori $v0, $0, 10  # system call code for exit
9  0x300c     syscall          # exit the program
```

instruction memory

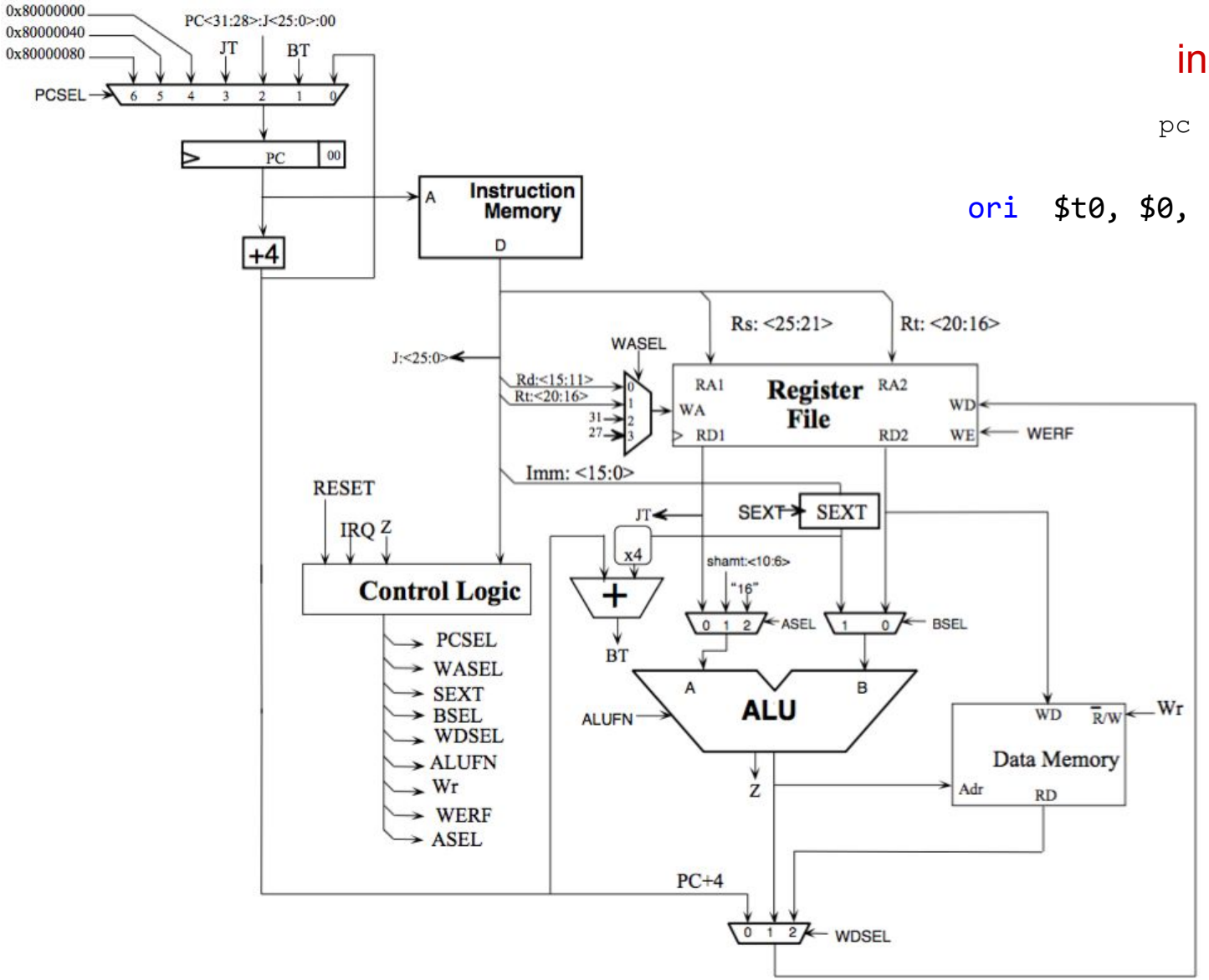
pc ->	0x3000	0x34080001
	0x3004	0x21090001
	0x3008	0x3402000a
	0x300c	0x0000000c
	0x3010	
	0x3014	
	0x3018	
	...	

registers

$t0	
$t1	
pc	0x3000

ORI Instruction

The program counter (PC) points to the 1st instruction



instruction memory

pc ->	0x3000	0x34080001
	0x3004	0x21090001
	0x3008	0x3402000a
	0x300c	0x0000000c
	0x3010	
	0x3014	
	0x3018	
	...	

```
ori  $t0, $0, 1
```

registers

$t0	
$t1	
pc	0x3000

MIPS Reference Data Card ("Green Card") 1. Pull along perforation to separate card 2. Fold bottom side (columns 3 and 4) together

MIPS Reference Data

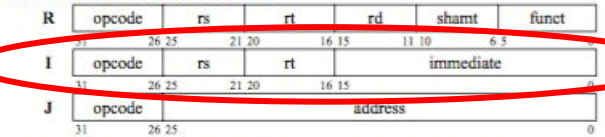
①

CORE INSTRUCTION SET

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+8;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)	24_{hex}
Load Halfword Unsigned	lhu	I	R[rt]={16'b0,M[R[rs]+SignExtImm](15:0)}	(2)	25_{hex}
Load Linked	ll	I	R[rt] = M[R[rs]+SignExtImm]	(2,7)	30_{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 Unsig.	sltu	R	R[rd] = (R[rs] < R[rt]) ? 1 : 0	(6)	0 / $2b_{hex}$
Shift Left Logical	sll	R	R[rd] = R[rt] << shamt		0 / 00_{hex}
Shift Right Logical	srl	R	R[rd] = R[rt] >>> shamt		0 / 02_{hex}
Store Byte	sb	I	M[R[rs]+SignExtImm](7:0) = R[rt](7:0)	(2)	28_{hex}
Store Conditional	sc	I	M[R[rs]+SignExtImm] = R[rt]; R[rt] = (*atomic*) ? 1 : 0	(2,7)	38_{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{1b'0}, immediate }
(4) BranchAddr = { 14{immediate[15]}, immediate, 2'b0 }
(5) JumpAddr = { PC+4[31:28], address, 2'b0 }
(6) Operands considered unsigned numbers (vs. 2's comp.)
(7) Atomic test&set pair; R[rt] = 1 if pair atomic, 0 if not atomic

BASIC INSTRUCTION FORMATS

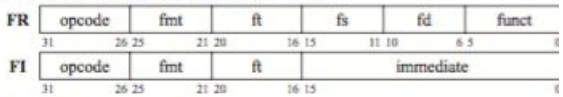


ARITHMETIC CORE INSTRUCTION SET

② OPCODE / FMT /FT / FUNCT (Hex)

NAME, MNEMONIC		FORMAT	OPERATION		
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]		10 /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
Shift Right Arith.	sra	R	R[rd] = R[rt] >> shamt		0/--/--/3
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/--/--/--

FLOATING-POINT INSTRUCTION FORMATS



PSEUDOINSTRUCTION SET

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]

REGISTER NAME, NUMBER, USE, CALL CONVENTION

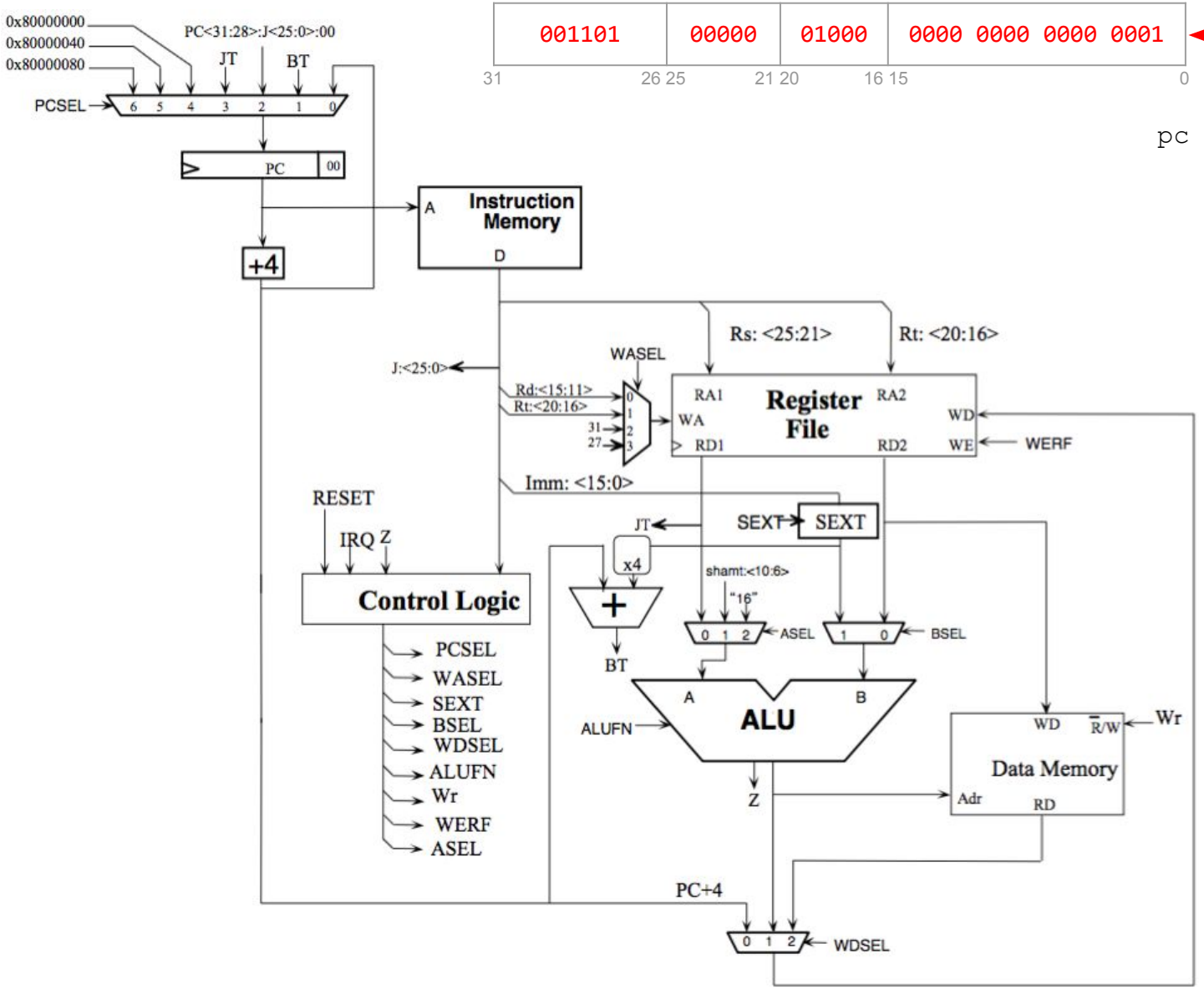
NAME	NUMBER	USE	PRESERVED ACROSS A CALL?
$zero	0	The Constant Value 0	N.A.
$at	1	Assembler Temporary	No
$v0-$v1	2-3	Values for Function Results and Expression Evaluation	No
$a0-$a3	4-7	Arguments	No
$t0-$t7	8-15	Temporaries	No
$s0-$s7	16-23	Saved Temporaries	Yes
$t8-$t9	24-25	Temporaries	No
$k0-$k1	26-27	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

```
ori  $t0, $0,  1
ori  $rt, $rs, imm   ->  R[rt] = R[rs] | signext(imm)
```

I-type	opcode	rs	rt	immediate

001101	00000	01000	0000 0000 0000 0001
31 – 26	25 – 21	20 – 16	15 – 0

Instruction format



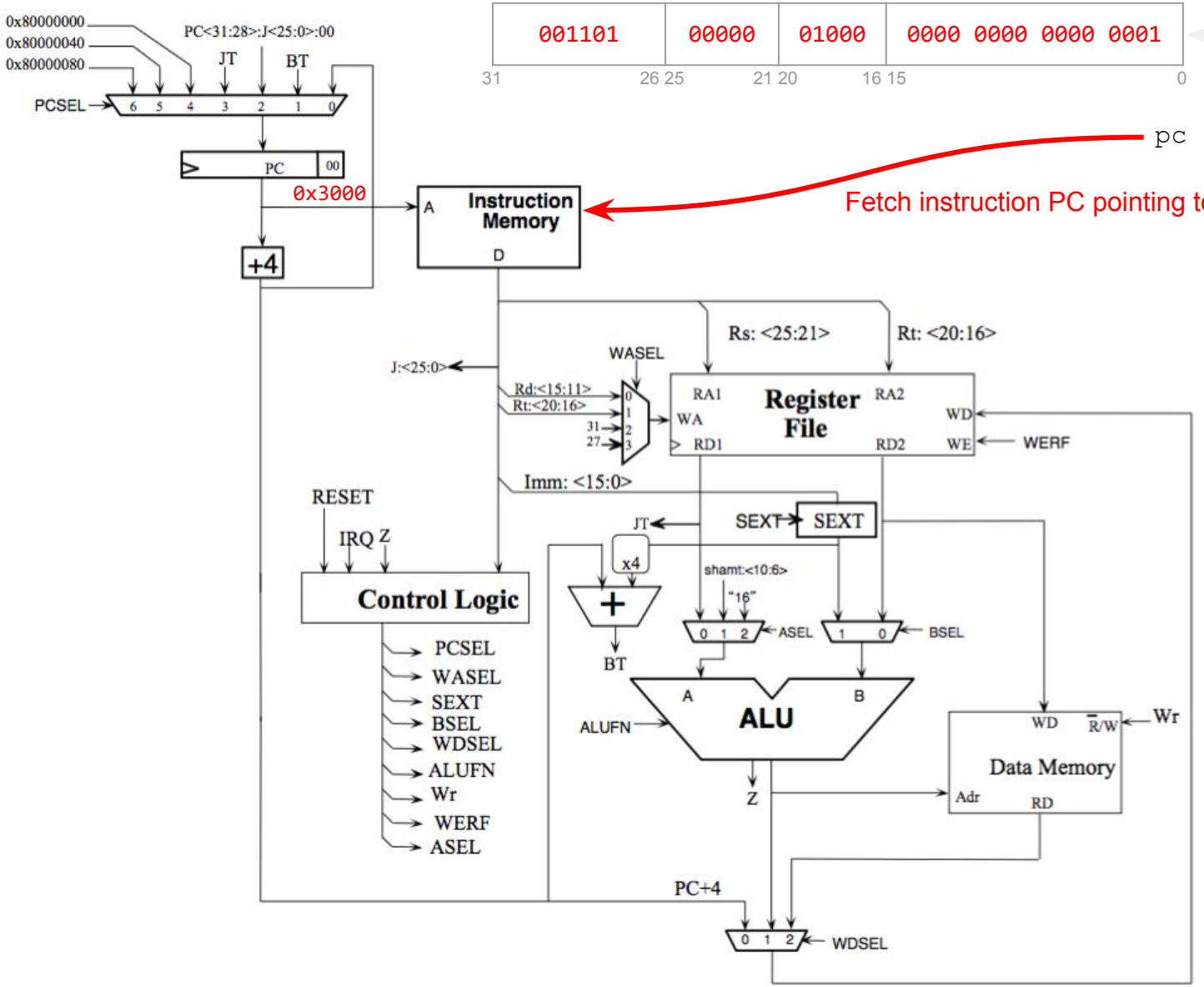
pc ->	0x3000	0x34080001
	0x3004	0x21090001
	0x3008	0x3402000a
	0x300c	0x0000000c
	0x3010	
	0x3014	
	0x3018	
	...	

registers

$t0	
$t1	
pc	0x3000

```
ori   $t0, $0,  1
ori   $rt, $rs, imm   ->   R[rt] = R[rs] | signext(imm)
```

I-type	opcode	rs	rt	immediate
	001101	00000	01000	0000 0000 0000 0001
	31	26 25	21 20	16 15 0



pc -> 0x3000 0x34080001

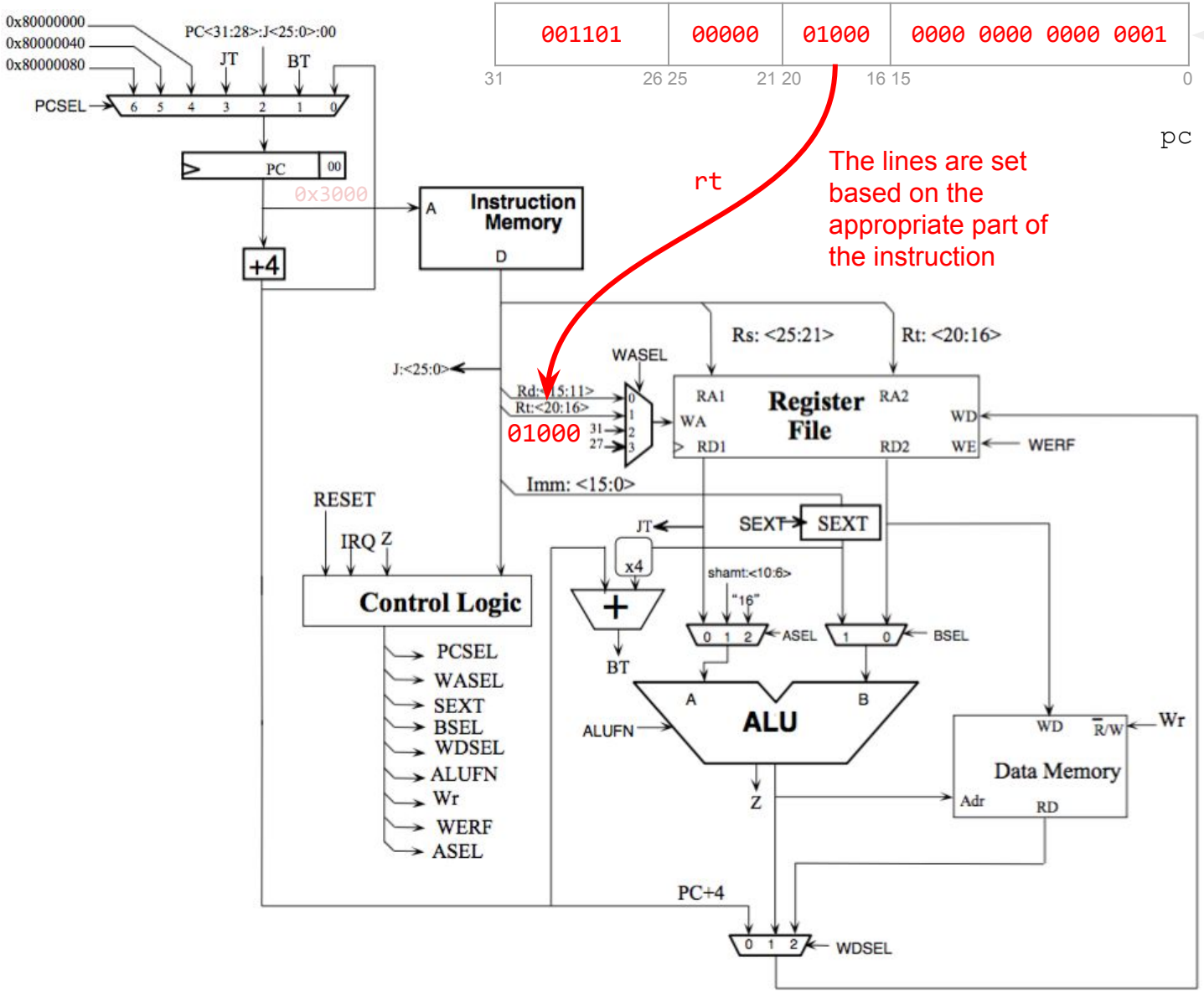
0x3000	0x34080001
0x3004	0x21090001
0x3008	0x3402000a
0x300c	0x0000000c
0x3010	
0x3014	
0x3018	
...	

registers

$t0	
$t1	
pc	0x3000

```
ori   $t0, $0,  1
ori   $rt, $rs, imm   ->   R[rt] = R[rs] | signext(imm)
```

I-type	opcode	rs	rt	immediate
	001101	00000	01000	0000 0000 0000 0001
	31 – 26	25 – 21	20 – 16	15 – 0



pc -> 0x3000	0x34080001
0x3004	0x21090001
0x3008	0x3402000a
0x300c	0x0000000c
0x3010	
0x3014	
0x3018	
...	

registers

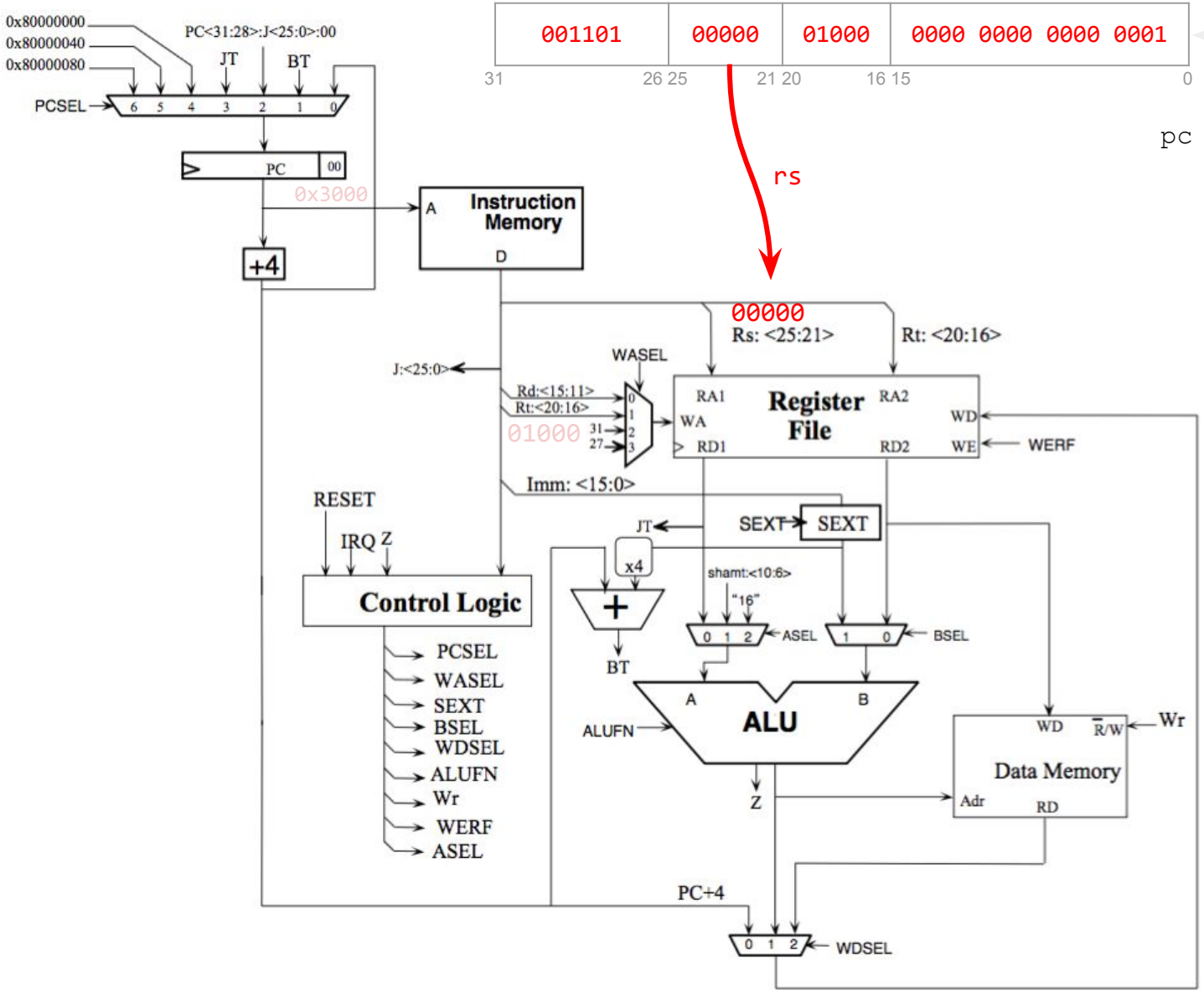
$t0	
$t1	
pc	0x3000

```
ori  $t0, $0,  1
ori  $rt, $rs, imm   ->  R[rt] = R[rs] | signext(imm)
```

I-type

opcode	rs	rt	immediate
001101	00000	01000	0000 0000 0000 0001
31 – 26	25 – 21	20 – 16	15 – 0

rs



pc -> 0x3000

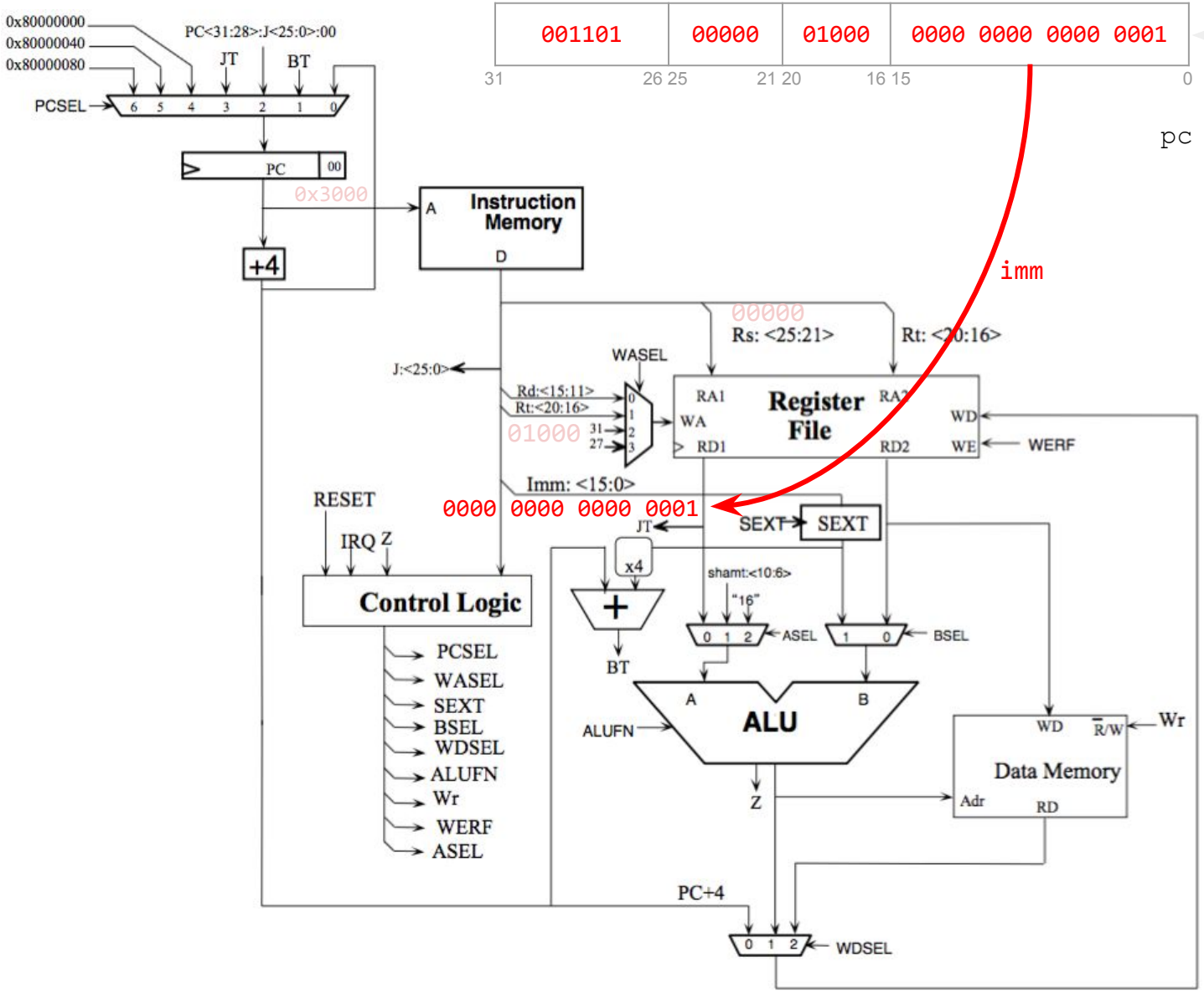
0x3000	0x34080001
0x3004	0x21090001
0x3008	0x3402000a
0x300c	0x0000000c
0x3010	
0x3014	
0x3018	
...	

registers

$t0	
$t1	
pc	0x3000

```
ori  $t0, $0,  1
ori  $rt, $rs, imm   ->  R[rt] = R[rs] | signext(imm)
```

I-type	opcode	rs	rt	immediate
	001101	00000	01000	0000 0000 0000 0001
	31 — 26	25 — 21	20 — 16	15 — 0

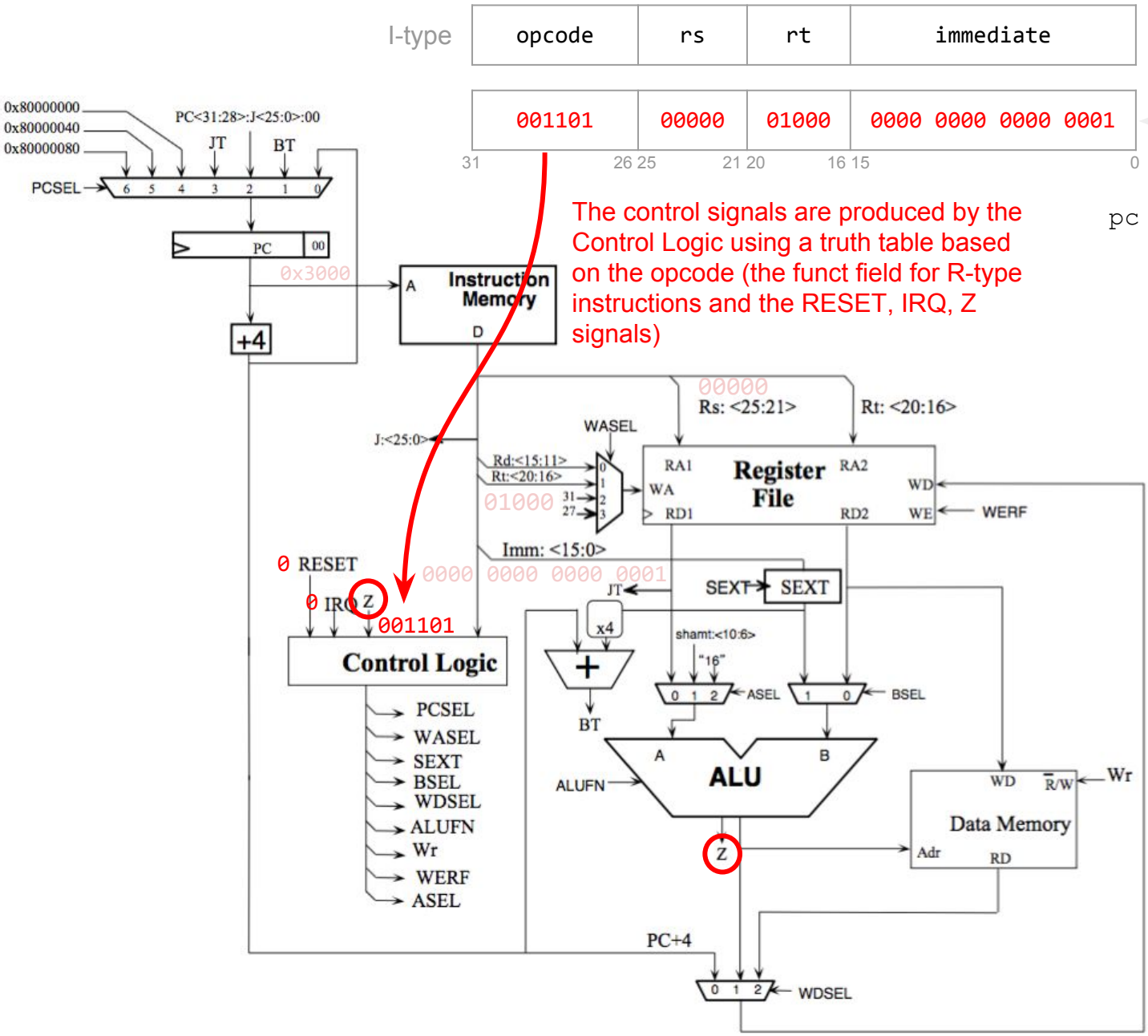


pc ->	0x3000	0x34080001
	0x3004	0x21090001
	0x3008	0x3402000a
	0x300c	0x0000000c
	0x3010	
	0x3014	
	0x3018	
	...	

registers

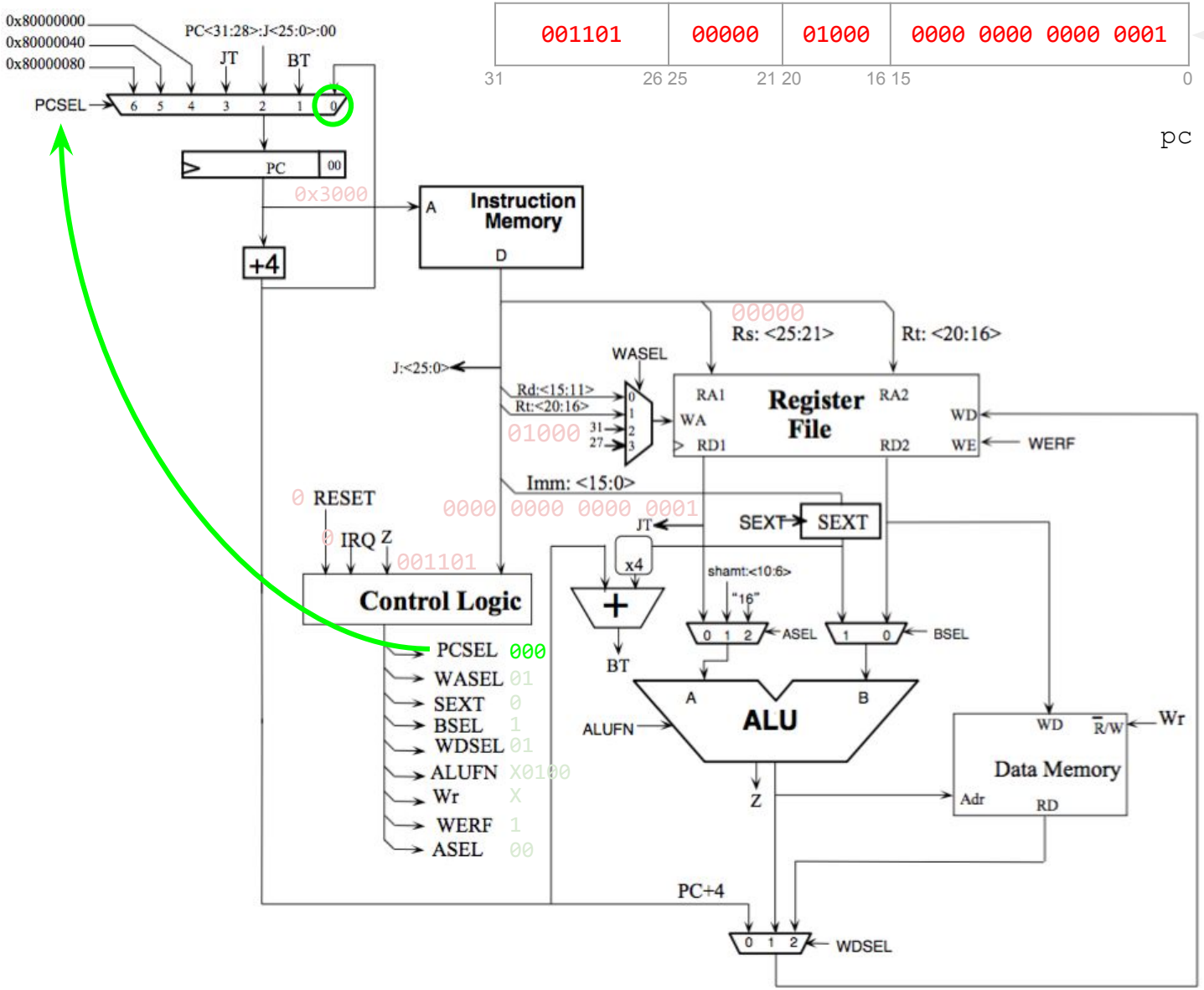
$t0	
$t1	
pc	0x3000

```
ori  $t0, $0,  1
ori  $rt, $rs, imm   ->  R[rt] = R[rs] | signext(imm)
```



```
ori   $t0, $0,  1
ori   $rt, $rs, imm   ->   R[rt] = R[rs] | signext(imm)
```

I-type	opcode	rs	rt	immediate
	001101	00000	01000	0000 0000 0000 0001
	31 – 26	25 – 21	20 – 16	15 – 0



pc -> 0x3000

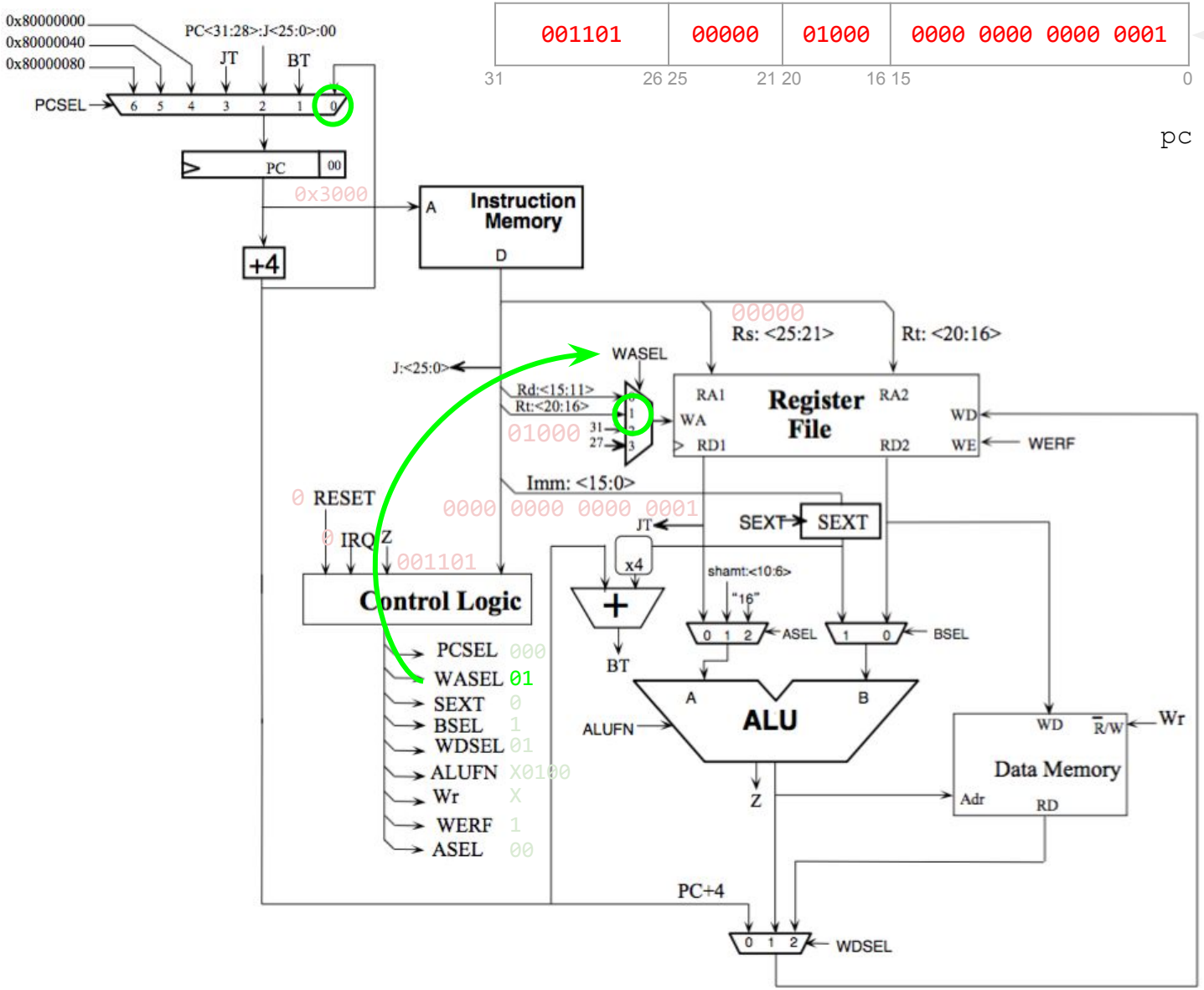
0x3000	0x34080001
0x3004	0x21090001
0x3008	0x3402000a
0x300c	0x0000000c
0x3010	
0x3014	
0x3018	
...	

registers

$t0	
$t1	
pc	0x3000

```
ori   $t0, $0,  1
ori   $rt, $rs, imm   ->   R[rt] = R[rs] | signext(imm)
```

I-type	opcode	rs	rt	immediate
	001101	00000	01000	0000 0000 0000 0001
	31 — 26	25 — 21	20 — 16	15 — 0



pc -> 0x3000

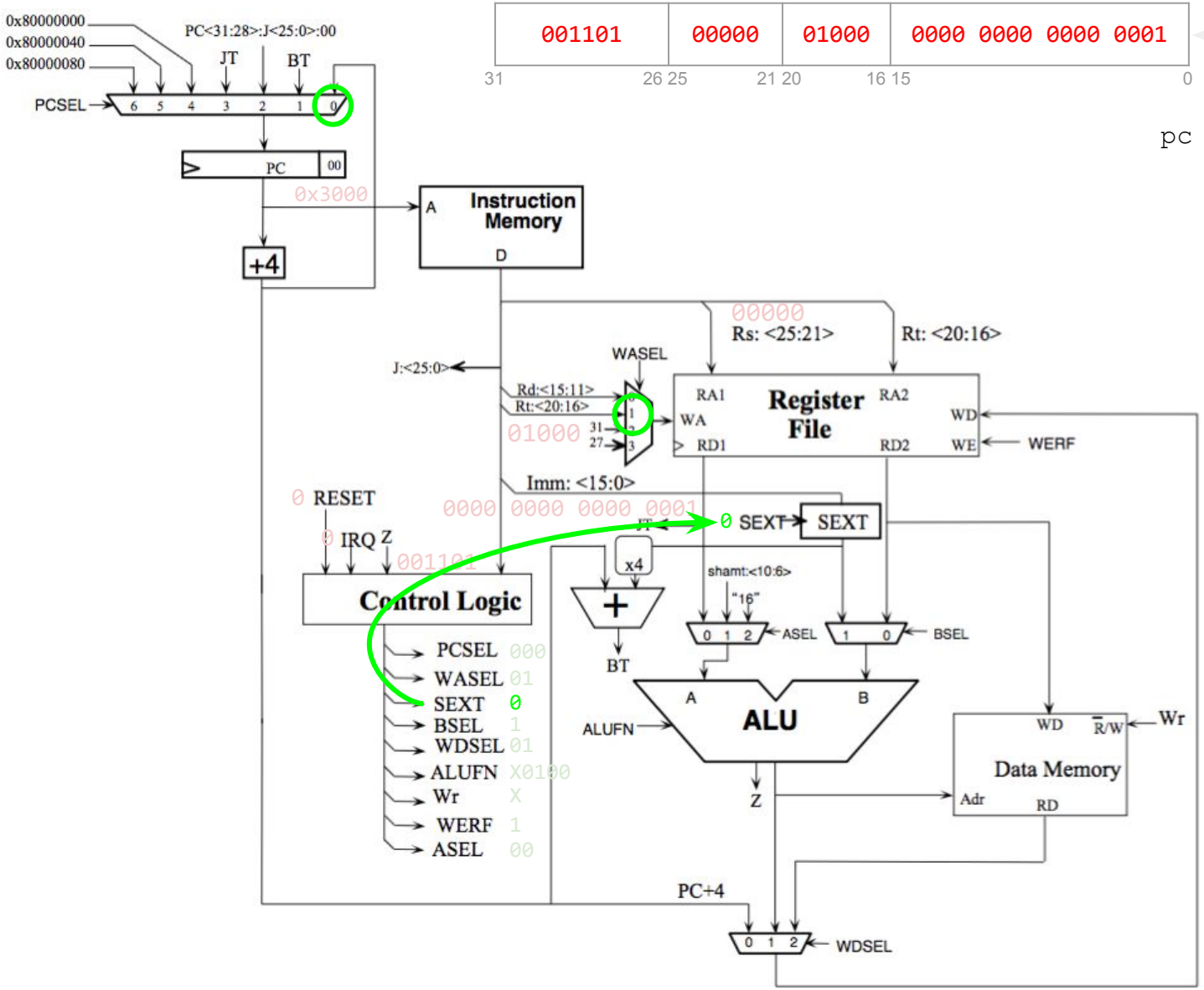
0x3000	0x34080001
0x3004	0x21090001
0x3008	0x3402000a
0x300c	0x0000000c
0x3010	
0x3014	
0x3018	
...	

registers

$t0	
$t1	
pc	0x3000

```
ori   $t0, $0,  1
ori   $rt, $rs, imm   ->   R[rt] = R[rs] | signext(imm)
```

I-type	opcode	rs	rt	immediate
	001101	00000	01000	0000 0000 0000 0001
	31 – 26	25 – 21	20 – 16	15 – 0



pc -> 0x3000

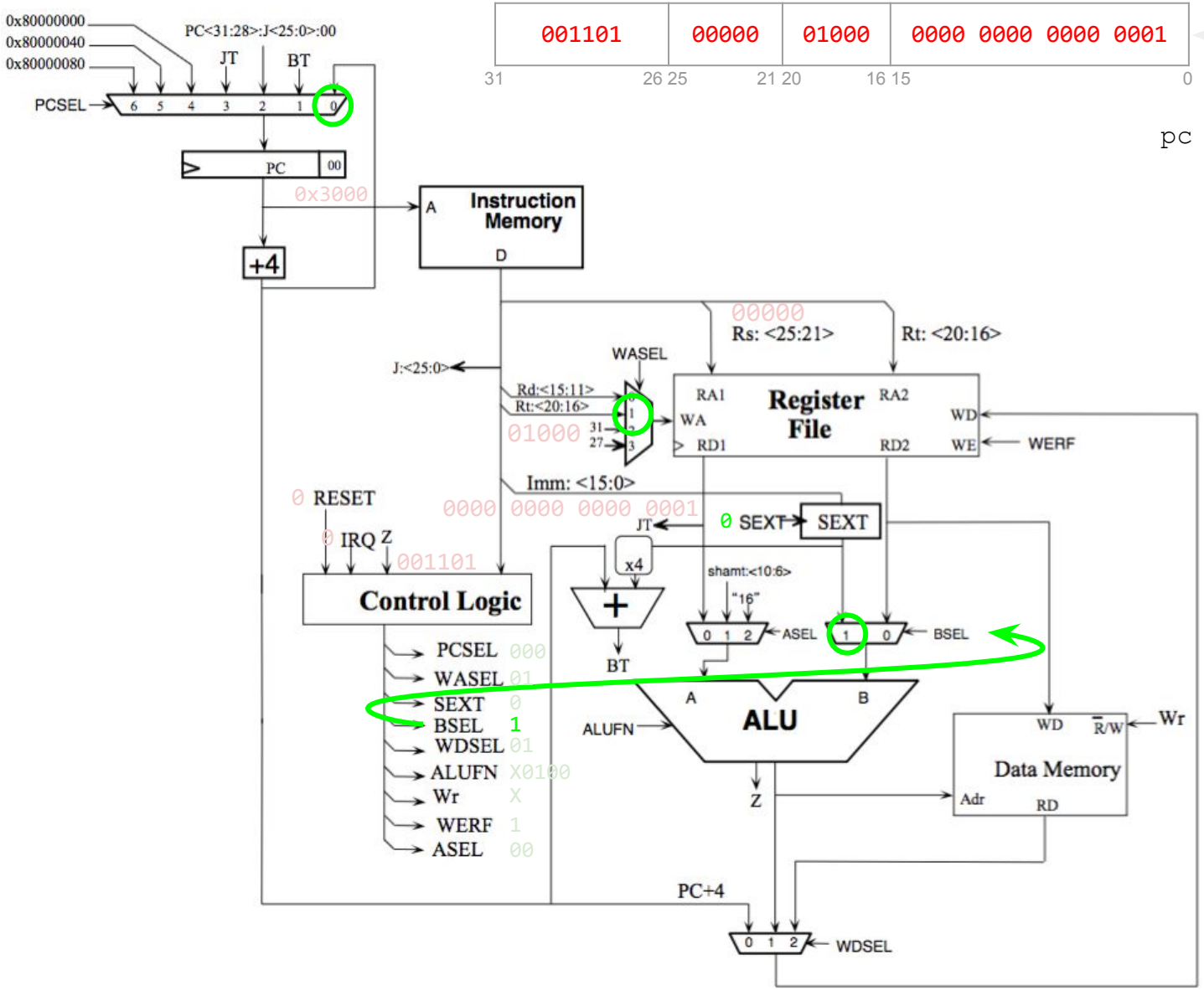
0x3000	0x34080001
0x3004	0x21090001
0x3008	0x3402000a
0x300c	0x0000000c
0x3010	
0x3014	
0x3018	
...	

registers

$t0	
$t1	
pc	0x3000

```
ori  $t0, $0,  1
ori  $rt, $rs, imm   ->  R[rt] = R[rs] | signext(imm)
```

I-type	opcode	rs	rt	immediate
	001101	00000	01000	0000 0000 0000 0001
	31 — 26	25 — 21	20 — 16	15 — 0



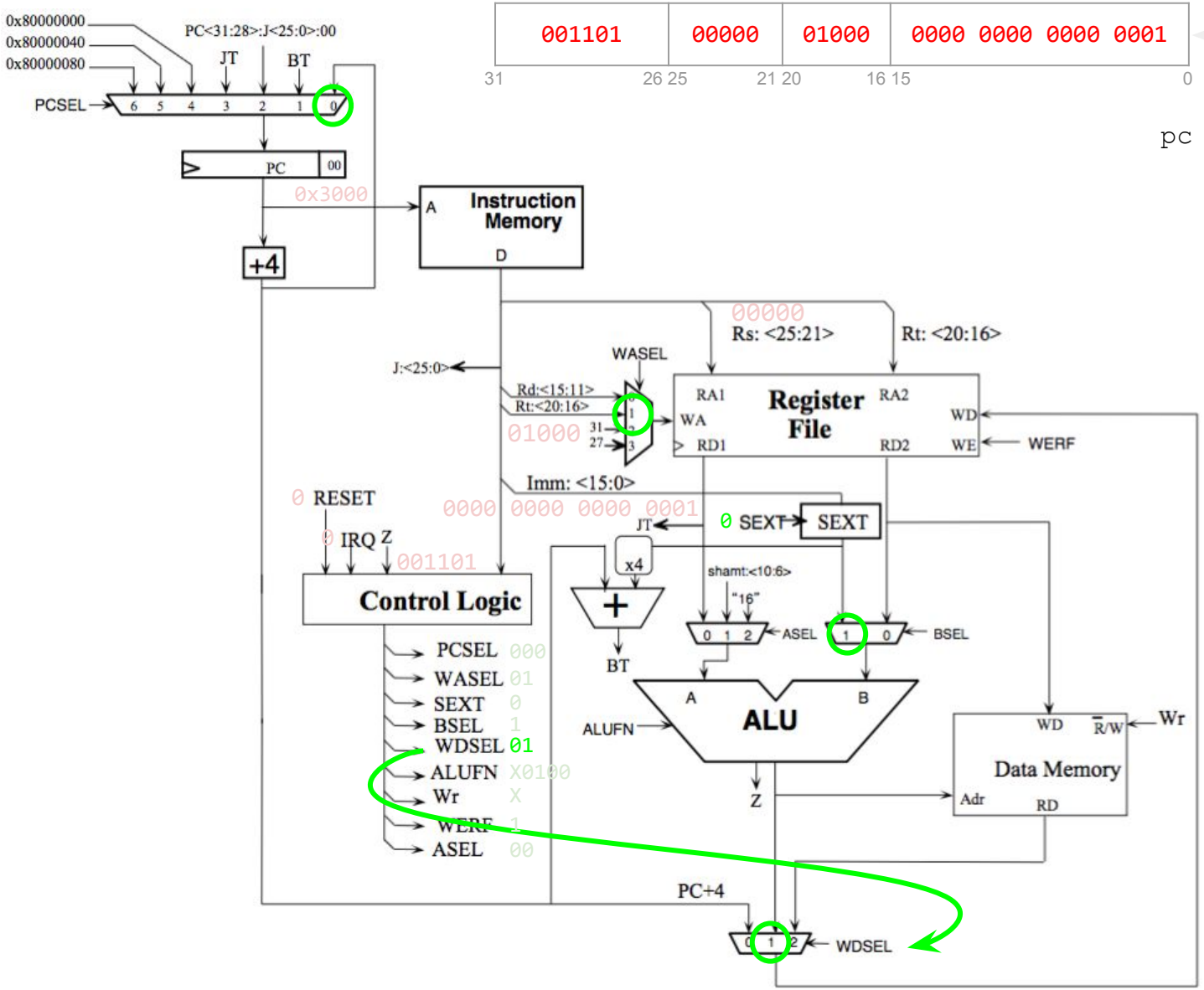
pc ->	
0x3000	0x34080001
0x3004	0x21090001
0x3008	0x3402000a
0x300c	0x0000000c
0x3010	
0x3014	
0x3018	
...	

registers

$t0	
$t1	
pc	0x3000

```
ori  $t0, $0,  1
ori  $rt, $rs, imm   ->  R[rt] = R[rs] | signext(imm)
```

I-type	opcode	rs	rt	immediate
	001101	00000	01000	0000 0000 0000 0001
	31 – 26	25 – 21	20 – 16	15 – 0



pc -> 0x3000

0x3000	0x34080001
0x3004	0x21090001
0x3008	0x3402000a
0x300c	0x0000000c
0x3010	
0x3014	
0x3018	
...	

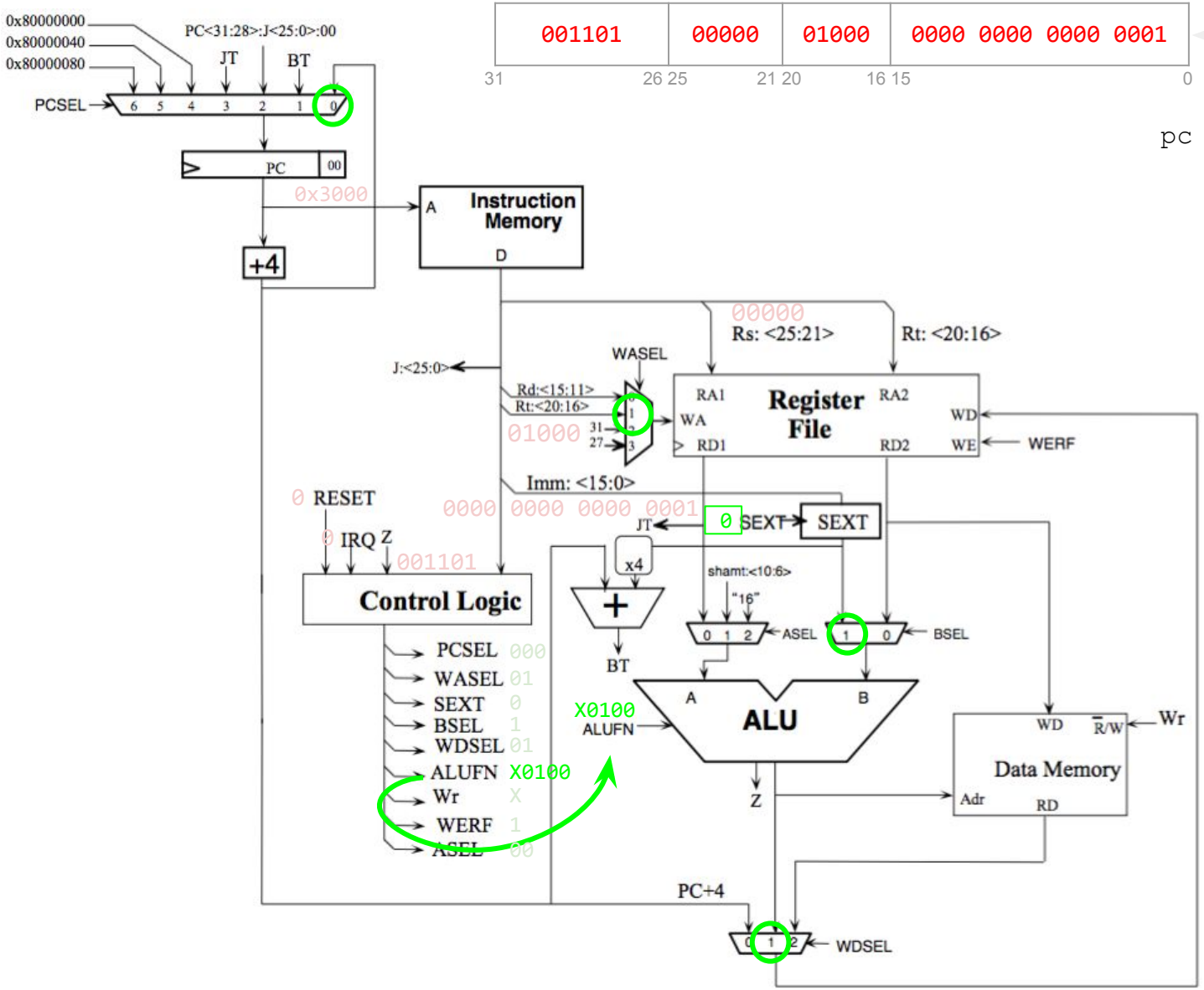
registers

$t0	
$t1	
pc	0x3000

```
ori  $t0, $0,  1
ori  $rt, $rs, imm   ->  R[rt] = R[rs] | signext(imm)
```

I-type

opcode	rs	rt	immediate
001101	00000	01000	0000 0000 0000 0001
31 – 26	25 – 21	20 – 16	15 – 0



pc -> 0x3000

0x3000	0x34080001
0x3004	0x21090001
0x3008	0x3402000a
0x300c	0x0000000c
0x3010	
0x3014	
0x3018	
...	

registers

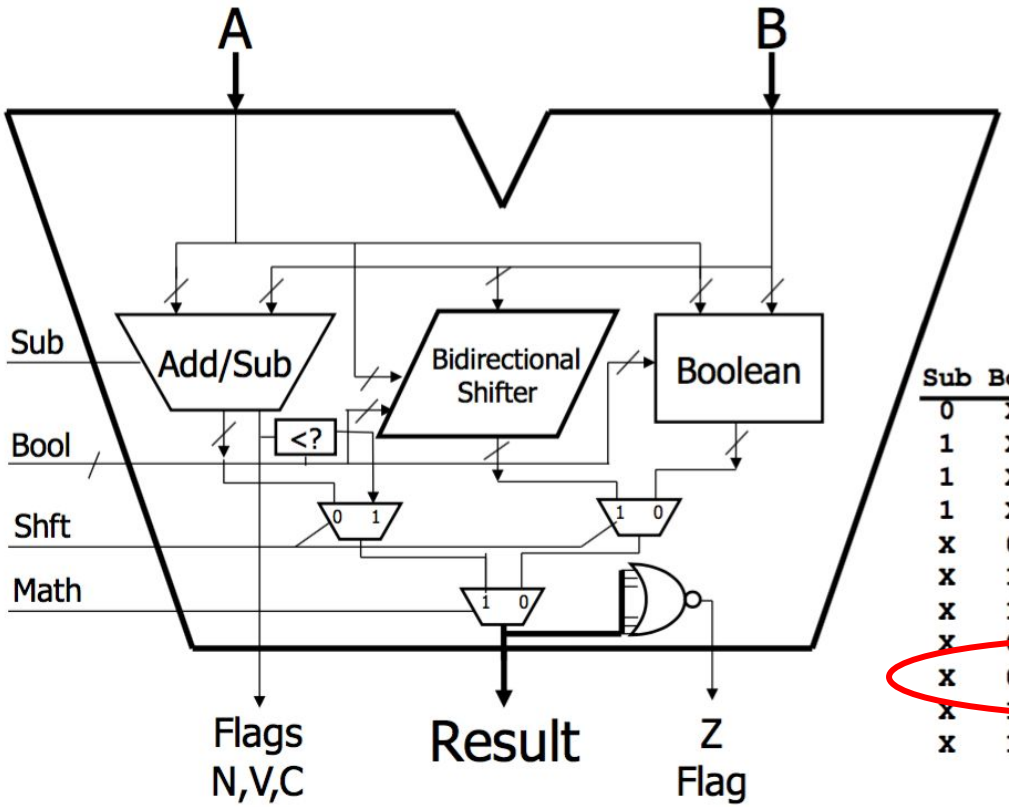
$t0	
$t1	
pc	0x3000

```
ori   $t0, $0,  1
ori   $rt, $rs, imm   ->   R[rt] = R[rs] | signext(imm)
```

pc -> 0x3000 0x34080001
0x3004 0x21090001
0x3008 0x3402000a
0x300c 0x0000000c
0x3010
0x3014
0x3018
...

registers
$t0
$t1
pc 0x3000

We covered the ALU functions earlier:

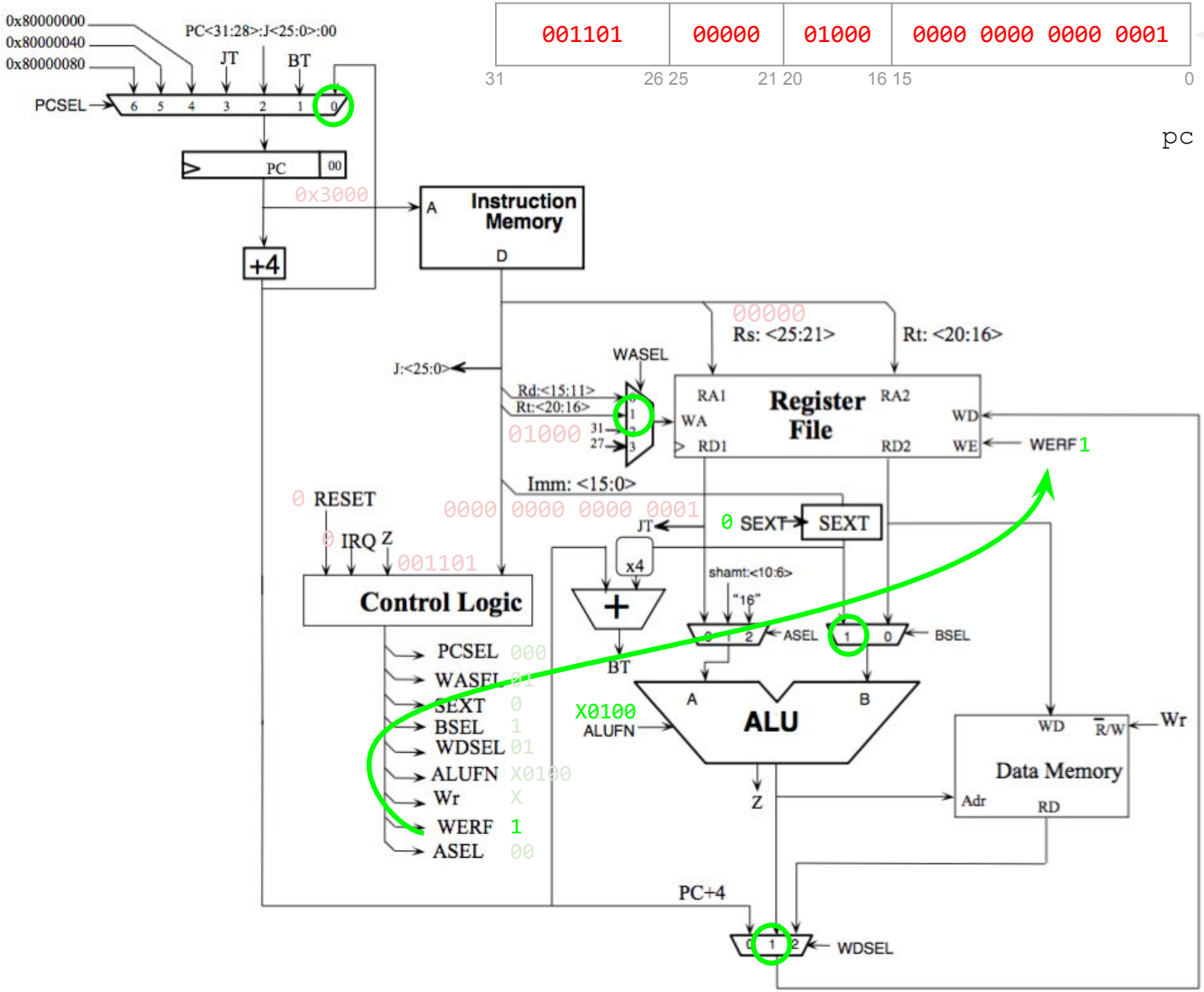


5-bit ALUFN

Sub	Bool	Shft	Math	OP
0	XX	0	1	A+B
1	XX	0	1	A-B
1	X0	1	1	A LT B
1	X1	1	1	A LTU B
X	00	1	0	B<<A
X	10	1	0	B>>A
X	11	1	0	B>>>A
X	00	0	0	A & B
X	01	0	0	A | B
X	10	0	0	A ^ B
X	11	0	0	~A | B

```
ori  $t0, $0,  1
ori  $rt, $rs, imm   ->  R[rt] = R[rs] | signext(imm)
```

I-type	opcode	rs	rt	immediate
	001101	00000	01000	0000 0000 0000 0001
	31 – 26	25 – 21	20 – 16	15 – 0



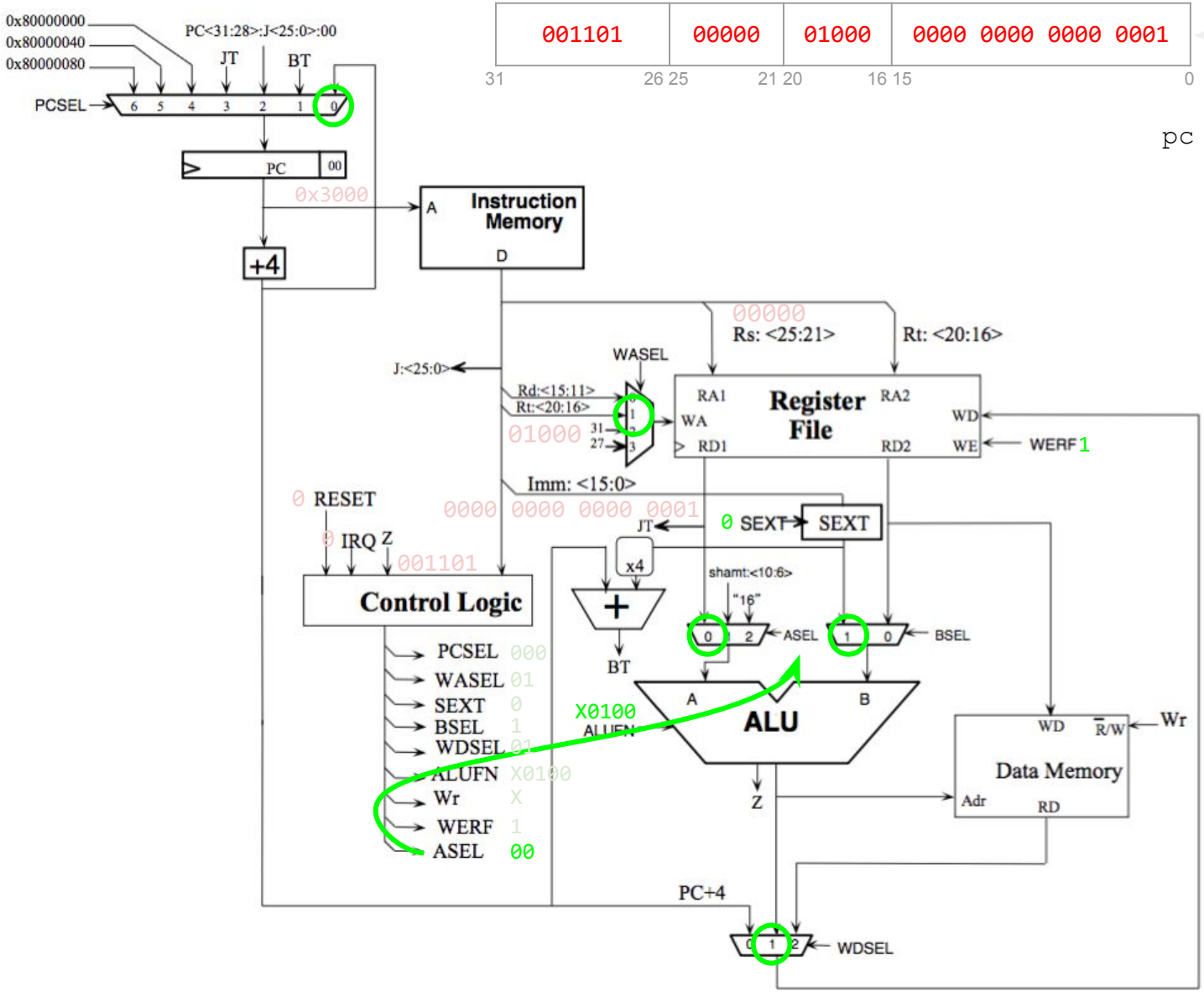
pc -> 0x3000	0x34080001
0x3004	0x21090001
0x3008	0x3402000a
0x300c	0x0000000c
0x3010	
0x3014	
0x3018	
...	

registers

$t0	
$t1	
pc	0x3000

```
ori   $t0, $0,  1
ori   $rt, $rs, imm   ->   R[rt] = R[rs] | signext(imm)
```

I-type	opcode	rs	rt	immediate
	001101	00000	01000	0000 0000 0000 0001
	31 — 26	25 — 21	20 — 16	15 — 0



pc -> 0x3000

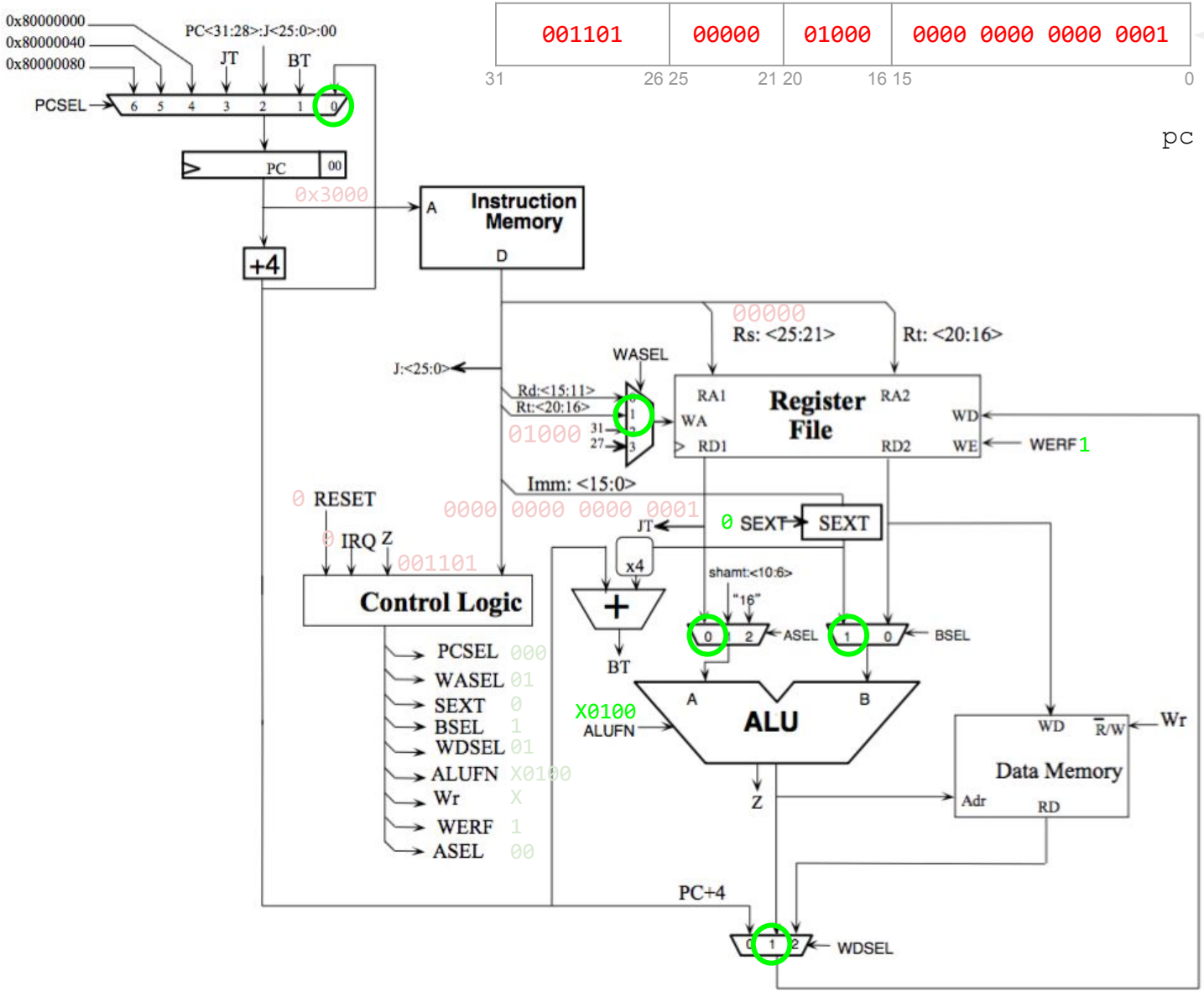
0x3000	0x34080001
0x3004	0x21090001
0x3008	0x3402000a
0x300c	0x0000000c
0x3010	
0x3014	
0x3018	
...	

registers

$t0	
$t1	
pc	0x3000

```
ori  $t0, $0,  1
ori  $rt, $rs, imm   ->  R[rt] = R[rs] | signext(imm)
```

I-type	opcode	rs	rt	immediate
	001101	00000	01000	0000 0000 0000 0001
	31 – 26	25 – 21	20 – 16	15 – 0



pc -> 0x3000

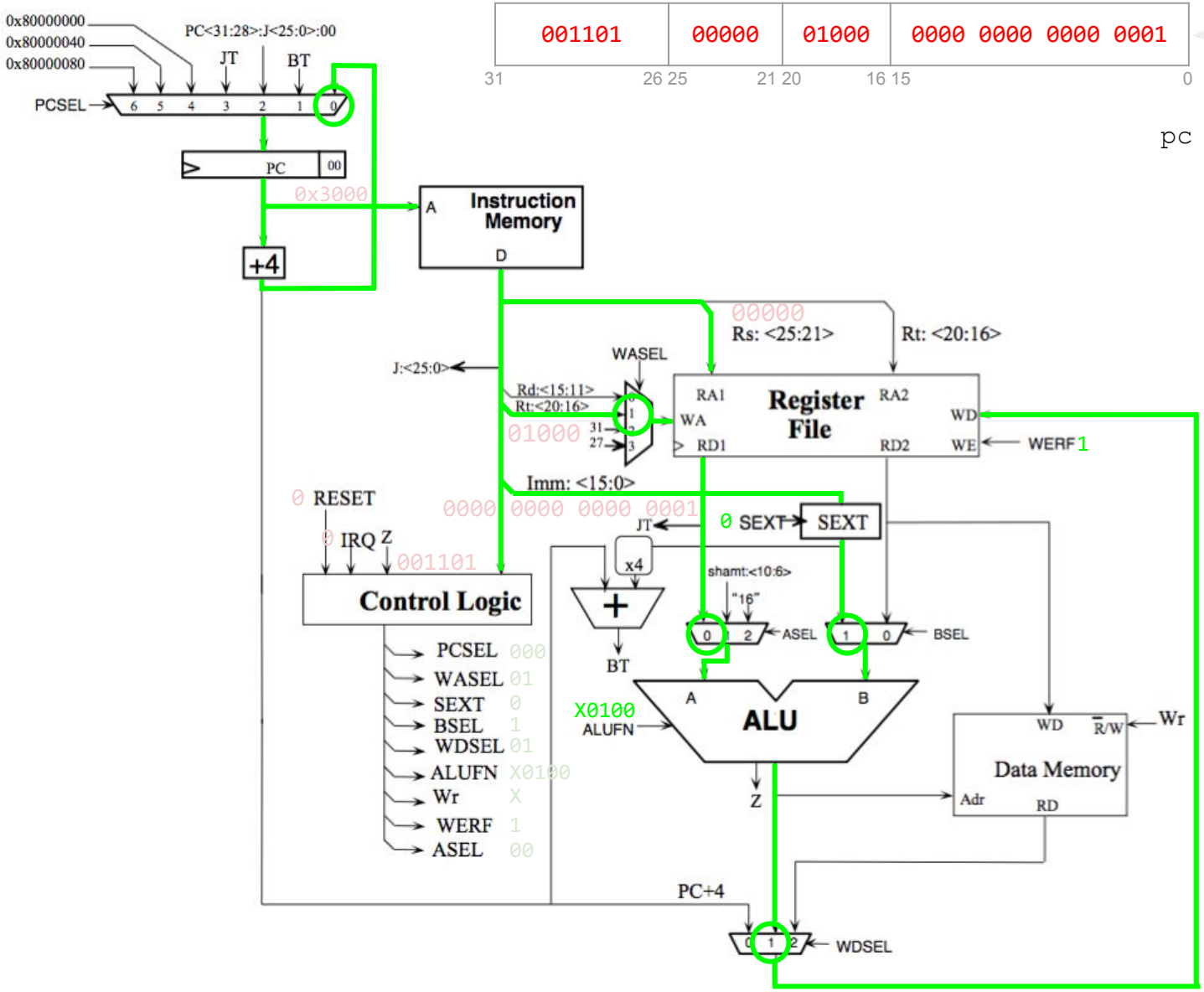
0x3000	0x34080001
0x3004	0x21090001
0x3008	0x3402000a
0x300c	0x0000000c
0x3010	
0x3014	
0x3018	
...	

registers

$t0	
$t1	
pc	0x3000

```
ori  $t0, $0,  1
ori  $rt, $rs, imm   ->  R[rt] = R[rs] | signext(imm)
```

I-type	opcode	rs	rt	immediate
	001101	00000	01000	0000 0000 0000 0001
	31 — 26	25 — 21	20 — 16	15 — 0



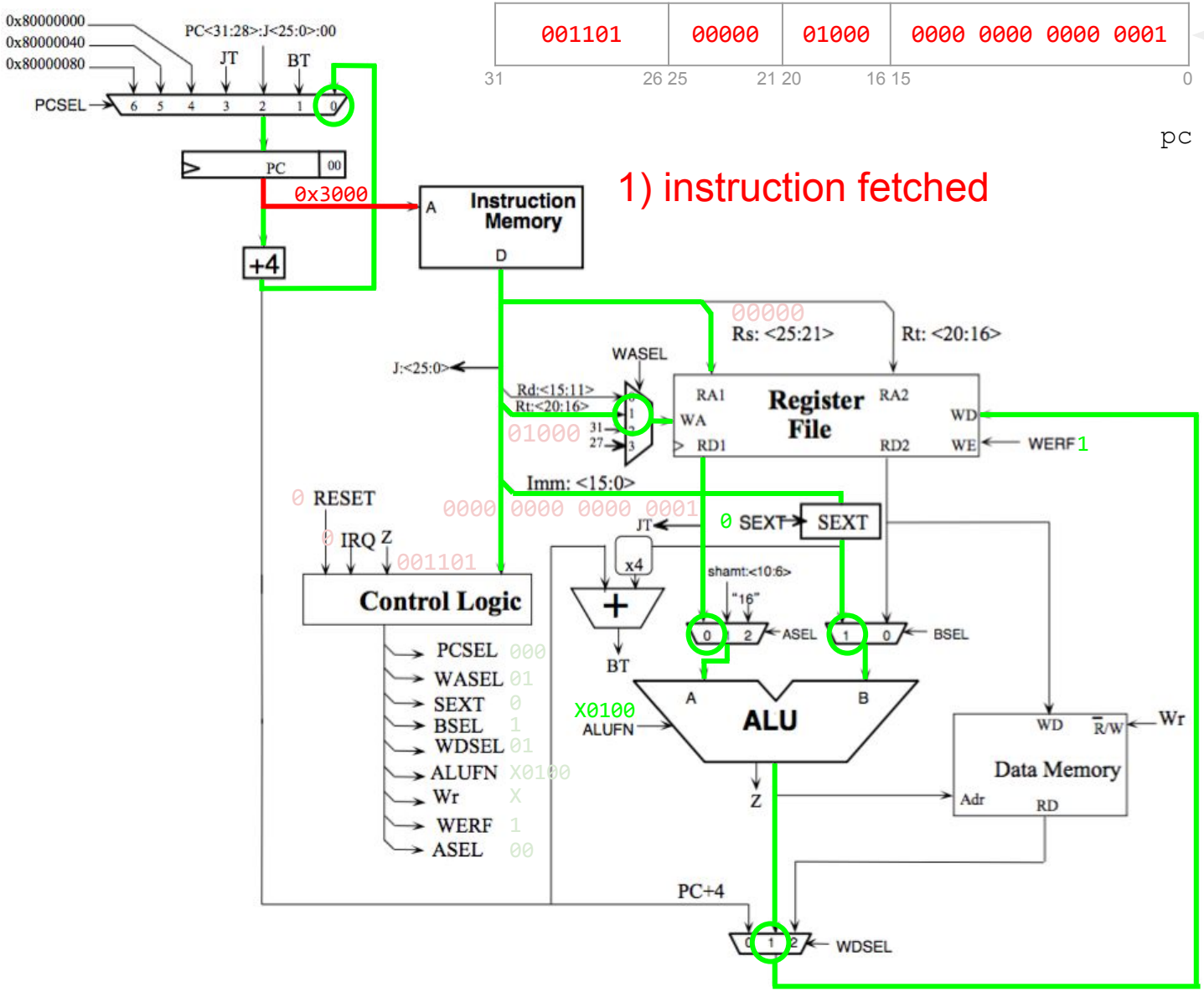
pc ->	0x3000	0x34080001
	0x3004	0x21090001
	0x3008	0x3402000a
	0x300c	0x0000000c
	0x3010	
	0x3014	
	0x3018	
	...	

registers

$t0	
$t1	
pc	0x3000

```
ori   $t0, $0,  1
ori   $rt, $rs, imm   ->   R[rt] = R[rs] | signext(imm)
```

I-type	opcode	rs	rt	immediate
	001101	00000	01000	0000 0000 0000 0001
	31 — 26	25 — 21	20 — 16	15 — 0



pc -> 0x3000

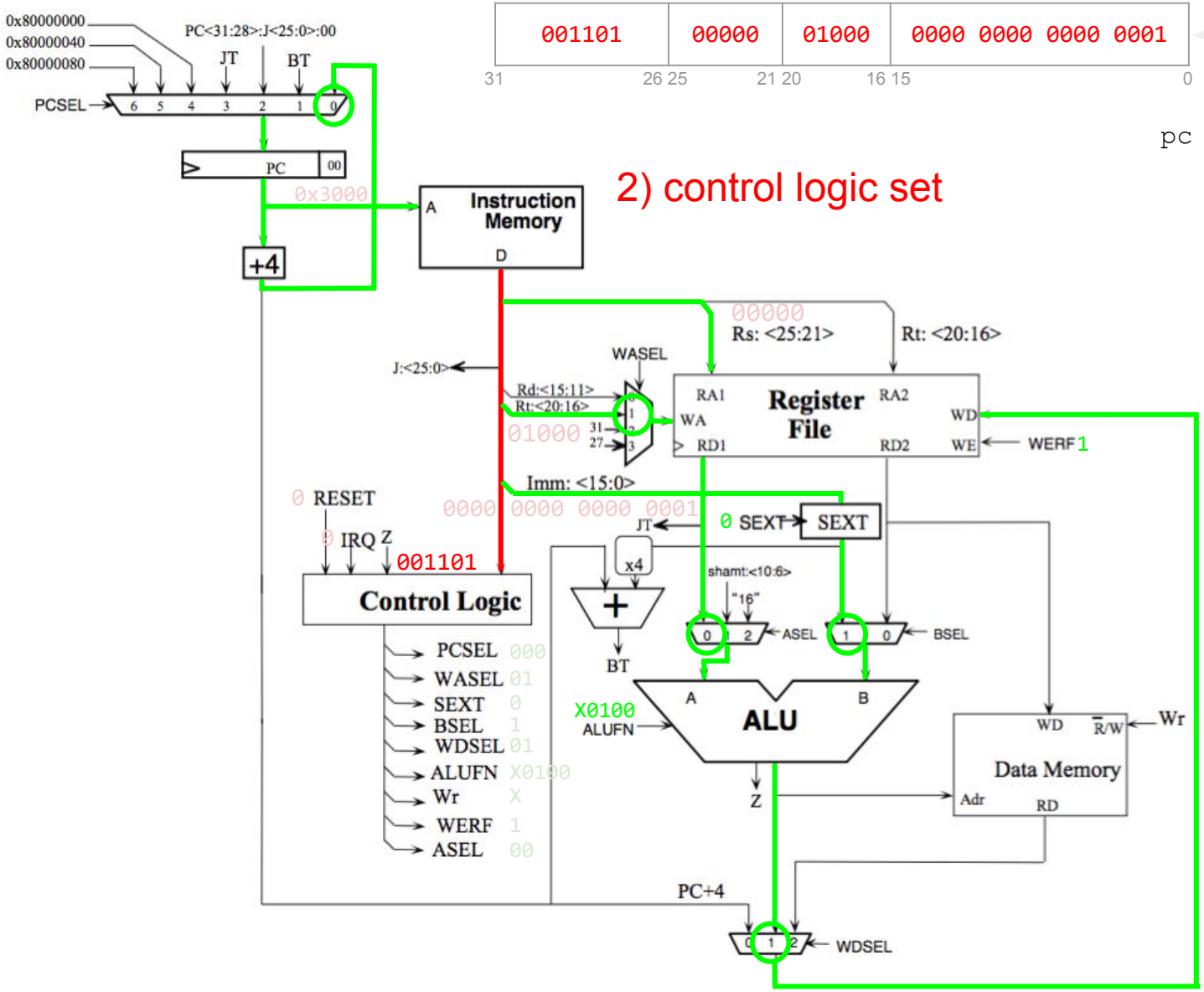
0x3000	0x34080001
0x3004	0x21090001
0x3008	0x3402000a
0x300c	0x0000000c
0x3010	
0x3014	
0x3018	
...	

registers

$t0	
$t1	
pc	0x3000

```
ori   $t0, $0,  1
ori   $rt, $rs, imm   ->   R[rt] = R[rs] | signext(imm)
```

I-type	opcode	rs	rt	immediate
	001101	00000	01000	0000 0000 0000 0001
	31 – 26	25 – 21	20 – 16	15 – 0



pc -> 0x3000

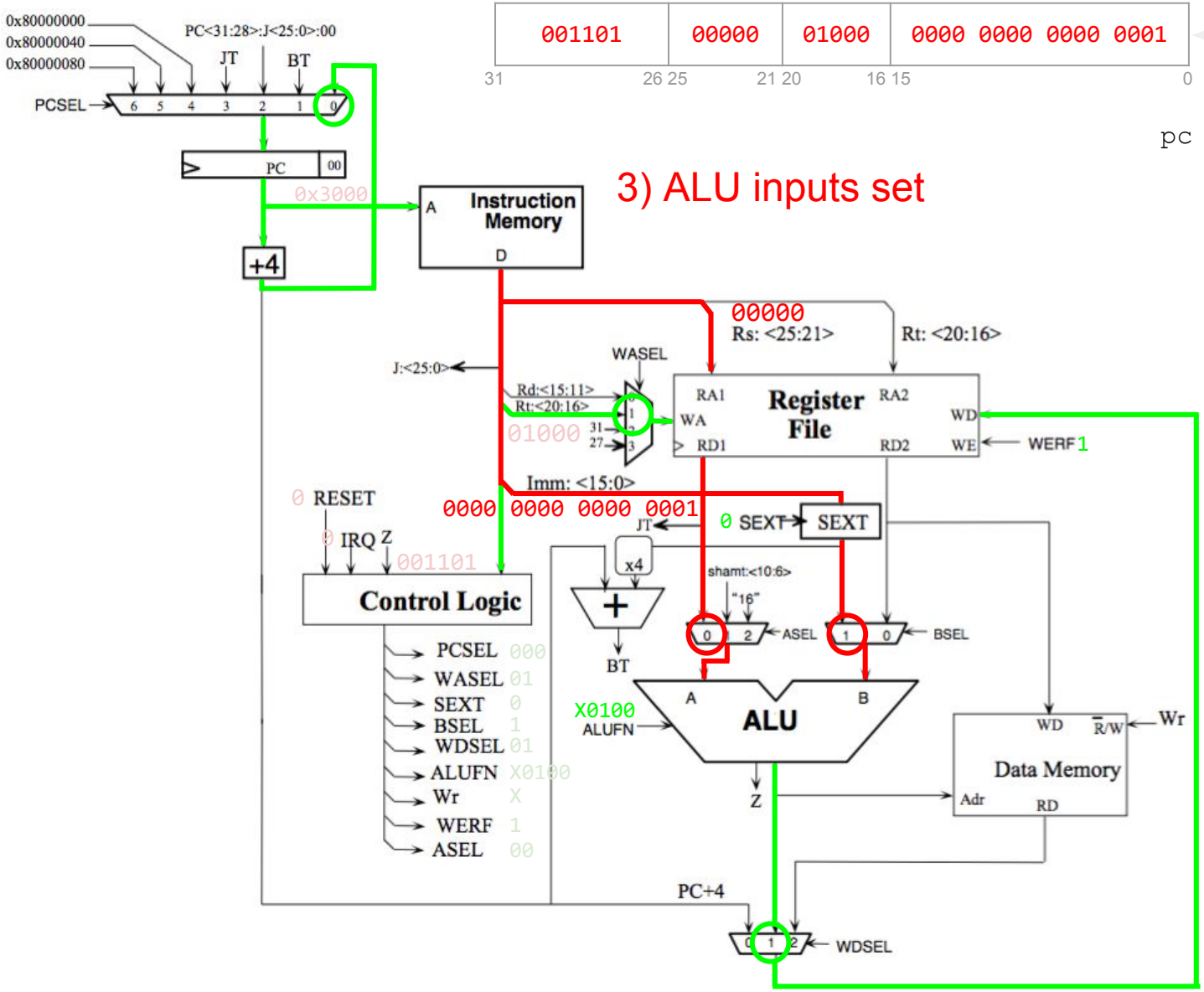
0x3000	0x34080001
0x3004	0x21090001
0x3008	0x3402000a
0x300c	0x0000000c
0x3010	
0x3014	
0x3018	
...	

registers

$t0	
$t1	
pc	0x3000

```
ori   $t0, $0,  1
ori   $rt, $rs, imm   ->   R[rt] = R[rs] | signext(imm)
```

I-type	opcode	rs	rt	immediate
	001101	00000	01000	0000 0000 0000 0001
	31 — 26	25 — 21	20 — 16	15 — 0



pc -> 0x3000

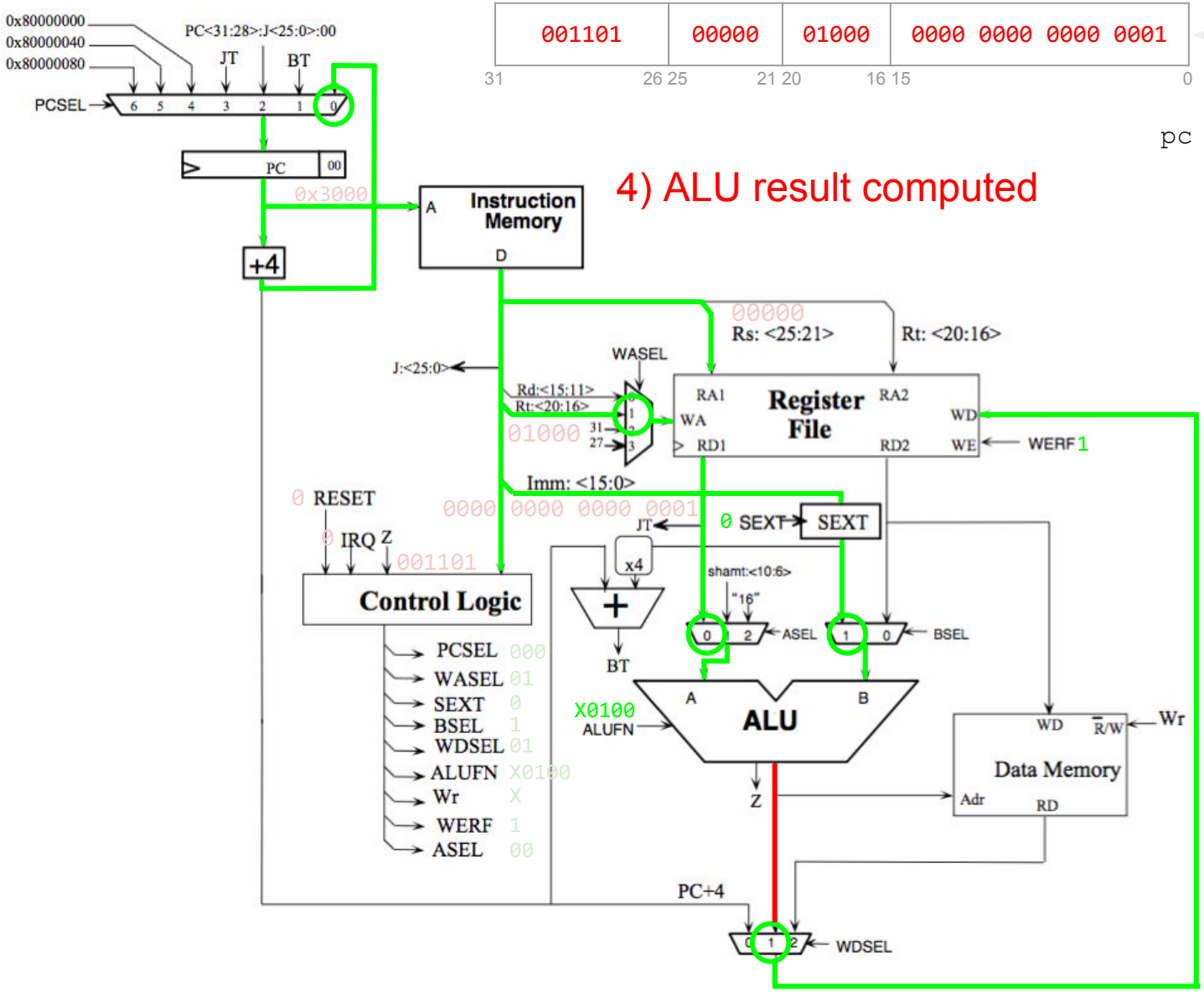
0x3000	0x34080001
0x3004	0x21090001
0x3008	0x3402000a
0x300c	0x0000000c
0x3010	
0x3014	
0x3018	
...	

registers

$t0	
$t1	
pc	0x3000

```
ori   $t0, $0,  1
ori   $rt, $rs, imm   ->   R[rt] = R[rs] | signext(imm)
```

I-type	opcode	rs	rt	immediate
	001101	00000	01000	0000 0000 0000 0001
	31 — 26	25 — 21	20 — 16	15 — 0



pc -> 0x3000

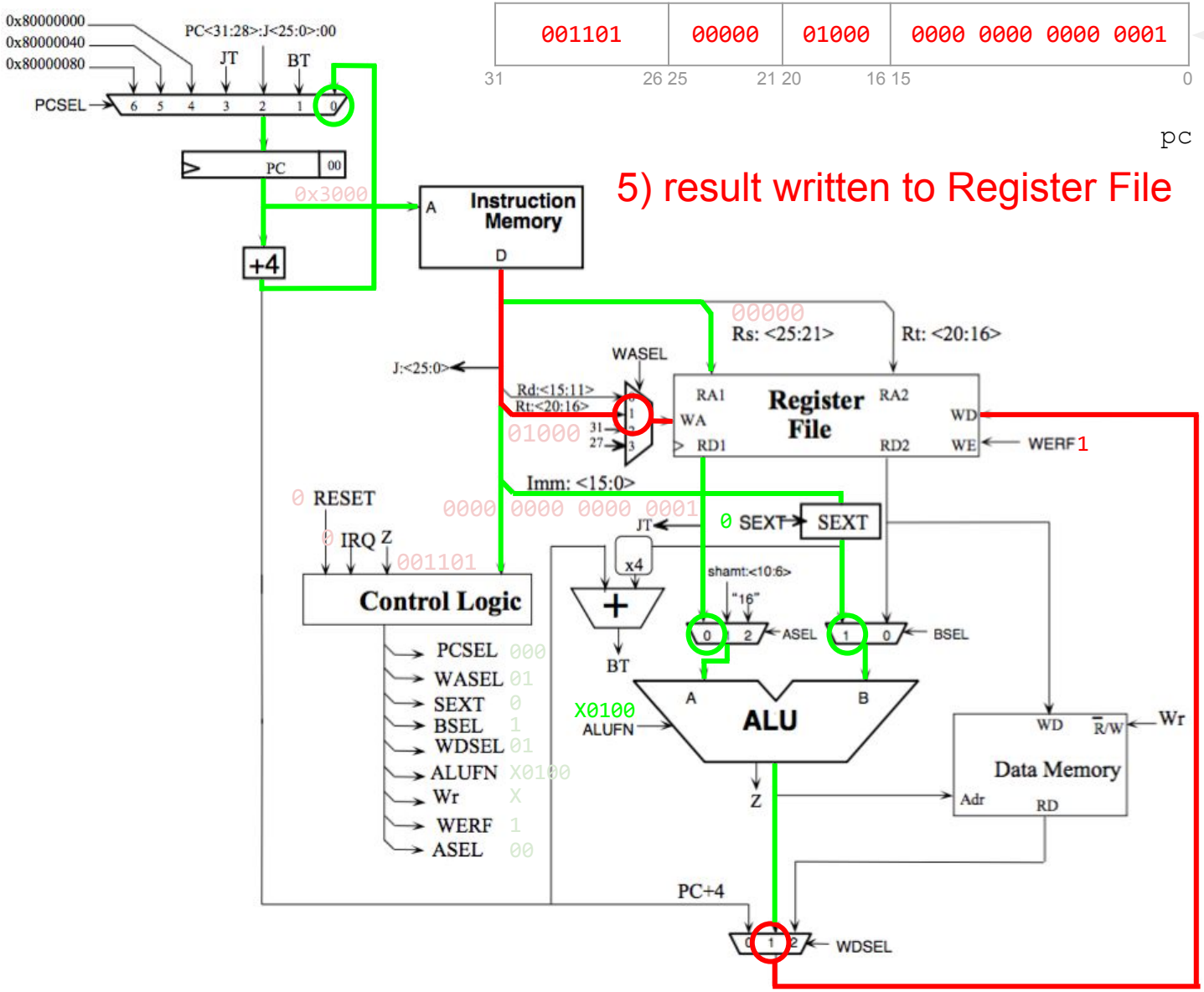
0x3000	0x34080001
0x3004	0x21090001
0x3008	0x3402000a
0x300c	0x0000000c
0x3010	
0x3014	
0x3018	
...	

registers

$t0	
$t1	
pc	0x3000

```
ori   $t0, $0,  1
ori   $rt, $rs, imm   ->   R[rt] = R[rs] | signext(imm)
```

I-type	opcode	rs	rt	immediate
	001101	00000	01000	0000 0000 0000 0001
	31 – 26	25 – 21	20 – 16	15 – 0



pc ->	
0x3000	0x34080001
0x3004	0x21090001
0x3008	0x3402000a
0x300c	0x0000000c
0x3010	
0x3014	
0x3018	
...	

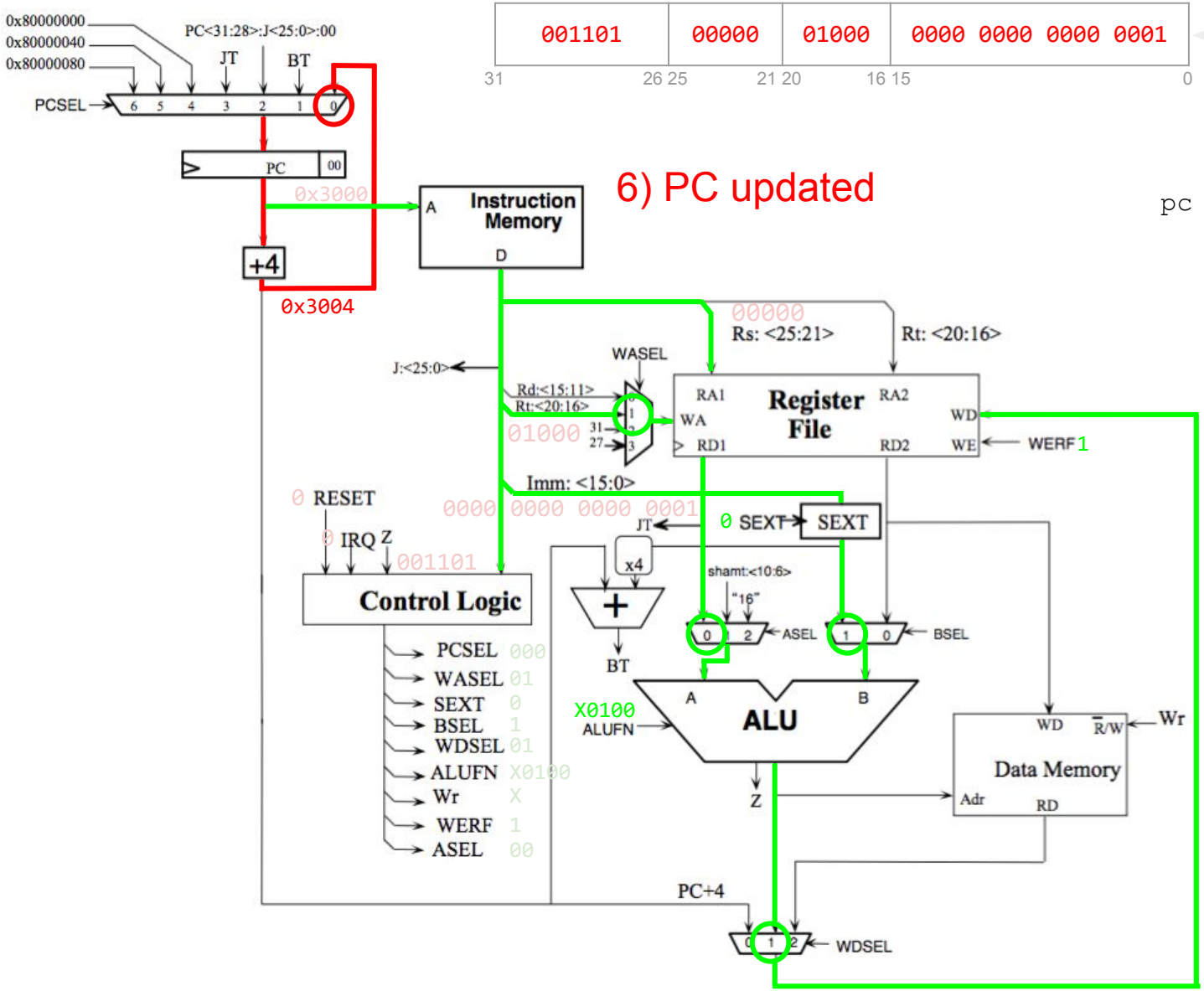
registers

$t0	0x00000001
$t1	
pc	0x3000

```
ori  $t0, $0,  1
ori  $rt, $rs, imm   ->  R[rt] = R[rs] | signext(imm)
```

I-type

opcode	rs	rt	immediate
001101	00000	01000	0000 0000 0000 0001
31 – 26	25 – 21	20 – 16	15 – 0



pc -> 0x3004

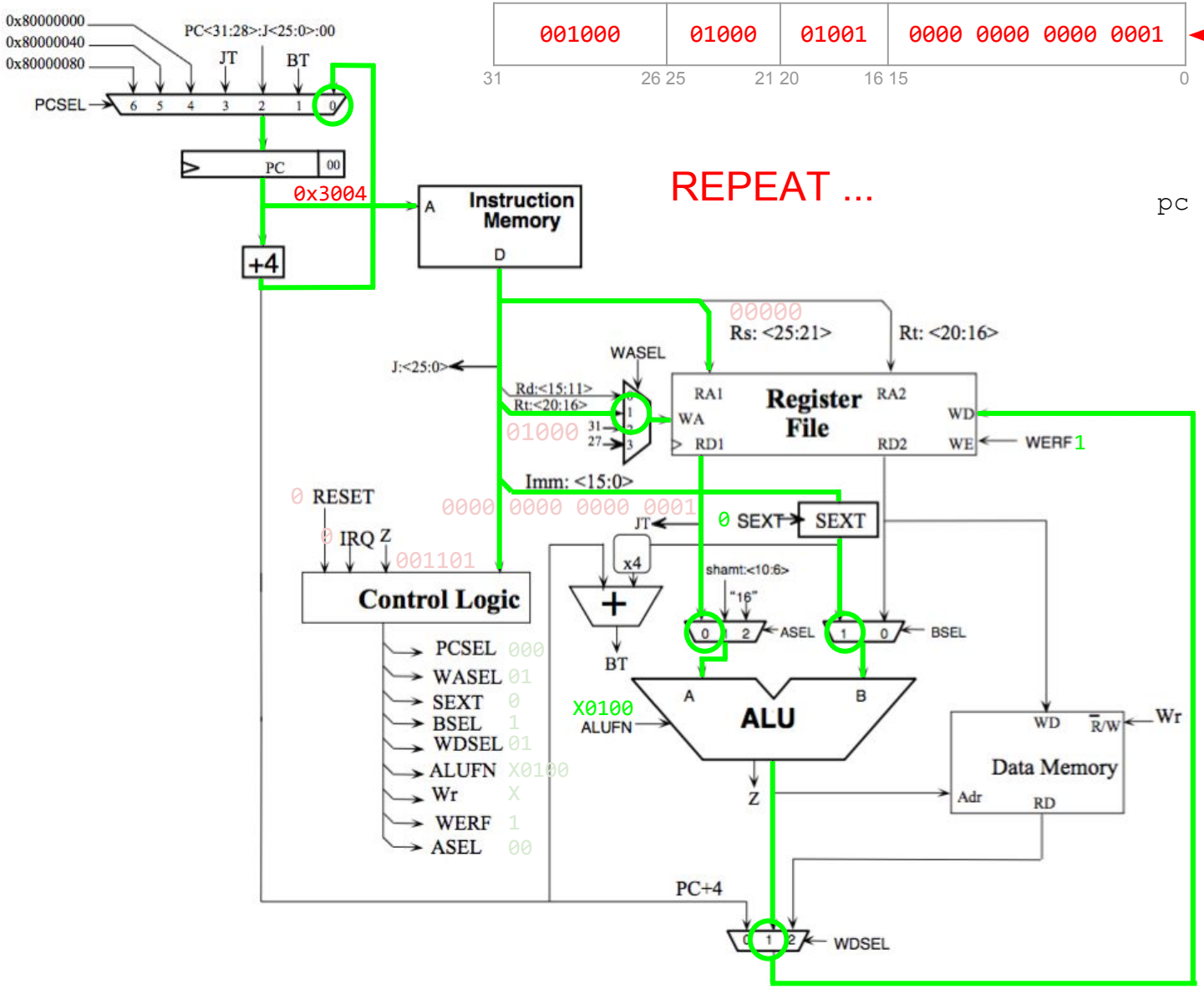
0x3000	0x34080001
0x3004	0x21090001
0x3008	0x3402000a
0x300c	0x0000000c
0x3010	
0x3014	
0x3018	
...	

registers

$t0	0x00000001
$t1	
pc	0x3004

```
addi $t1, $t0, 1
addi $rt, $rs, imm   ->   R[rt] = R[rs] + signext(imm)
```

I-type	opcode	rs	rt	immediate
	001000	01000	01001	0000 0000 0000 0001
	31 — 26	25 — 21	20 — 16	15 — 0



REPEAT ...

	0x3000	0x34080001
pc ->	0x3004	0x21090001
	0x3008	0x3402000a
	0x300c	0x0000000c
	0x3010	
	0x3014	
	0x3018	
	...	

registers

$t0	0x00000001
$t1	
pc	0x3004