CS/ECE 252
Exam 3 Review

11AM section

2015 November 20

Before we get started

Homework 7 due at beginning of lecture

Exam 3 on Monday, November 23rd, during class

Homework 8 will be released on/by Wednesday, November 25

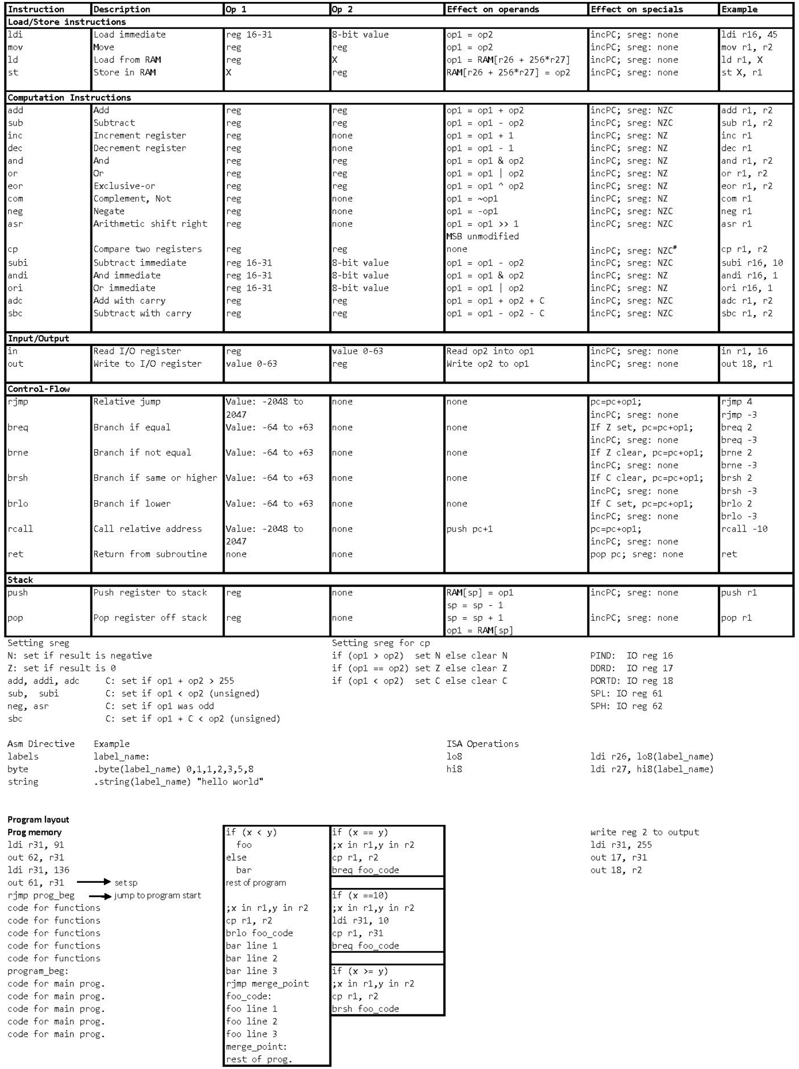
HW8 due on Friday, December 11th, at beginning of class

Quote of the day:

> “Success consists of going from failure to failure without loss of enthusiasm”
>
> -- Sir Winston Churchill
> 1874-1965

Today: Exam 3 Review

Cheatsheets 1-2: Chapters 5-6



Instruction	Description	Op 1	Op 2	Effect on operands	Effect on specials	Example
Load/Store instructions						
ldi	Load immediate	reg 16-31	8-bit value	op1 = op2	incPC; sreg: none	ldi r16, 45
mov	Move	reg	reg	op1 = op2	incPC; sreg: none	mov r1, r2
ld	Load from RAM	reg	X	op1 = RAM[r26 + 256*r27]	incPC; sreg: none	ld r1, X
st	Store in RAM	X	reg	RAM[r26 + 256*r27] = op2	incPC; sreg: none	st X, r1
Computation Instructions						
add	Add	reg	reg	op1 = op1 + op2	incPC; sreg: NZC	add r1, r2
sub	Subtract	reg	reg	op1 = op1 - op2	incPC; sreg: NZC	sub r1, r2
inc	Increment register	reg	none	op1 = op1 + 1	incPC; sreg: NZ	inc r1
dec	Decrement register	reg	none	op1 = op1 - 1	incPC; sreg: NZ	dec r1
and	And	reg	reg	op1 = op1 & op2	incPC; sreg: NZ	and r1, r2
or	Or	reg	reg	op1 = op1 | op2	incPC; sreg: NZ	or r1, r2
eor	Exclusive-or	reg	reg	op1 = op1 ^ op2	incPC; sreg: NZ	eor r1, r2
com	Complement, Not	reg	none	op1 = ~op1	incPC; sreg: NZ	com r1
neg	Negate	reg	none	op1 = -op1	incPC; sreg: NZC	neg r1
asr	Arithmetic shift right	reg	none	op1 = op1 >> 1 MSB unmodified	incPC; sreg: NZC	asr r1
cp	Compare two registers	reg	reg	none	incPC; sreg: NZC#	cp r1, r2
subi	Subtract immediate	reg 16-31	8-bit value	op1 = op1 - op2	incPC; sreg: NZC	subi r16, 10
andi	And immediate	reg 16-31	8-bit value	op1 = op1 & op2	incPC; sreg: NZ	andi r16, 1
ori	Or immediate	reg 16-31	8-bit value	op1 = op1 | op2	incPC; sreg: NZ	ori r16, 1
adc	Add with carry	reg	reg	op1 = op1 + op2 + C	incPC; sreg: NZC	adc r1, r2
sbc	Subtract with carry	reg	reg	op1 = op1 - op2 - C	incPC; sreg: NZC	sbc r1, r2
Input/Output						
in	Read I/O register	reg	value 0-63	Read op2 into op1	incPC; sreg: none	in r1, 16
out	Write to I/O register	value 0-63	reg	Write op2 to op1	incPC; sreg: none	out 18, r1
Control-Flow						
rjmp	Relative jump	Value: -2048 to 2047	none	none	pc=pc+op1; incPC; sreg: none	rjmp 4 rjmp -3
breq	Branch if equal	Value: -64 to +63	none	none	If Z set, pc=pc+op1; incPC; sreg: none	breq 2 breq -3
brne	Branch if not equal	Value: -64 to +63	none	none	If Z clear, pc=pc+op1; incPC; sreg: none	brne 2 brne -3
brsh	Branch if same or higher	Value: -64 to +63	none	none	If C clear, pc=pc+op1; incPC; sreg: none	brsh 2 brsh -3
brlo	Branch if lower	Value: -64 to +63	none	none	If C set, pc=pc+op1; incPC; sreg: none	brlo 2 brlo -3
rcall	Call relative address	Value: -2048 to 2047	none	push pc+1	pc=pc+op1; incPC; sreg: none	rcall -10
ret	Return from subroutine	none	none		pop pc; sreg: none	ret
Stack						
push	Push register to stack	reg	none	RAM[sp] = op1 sp = sp - 1	incPC; sreg: none	push r1
pop	Pop register off stack	reg	none	sp = sp + 1 op1 = RAM[sp]	incPC; sreg: none	pop r1

```
Setting sreg
N: set if result is negative
Z: set if result is 0
add, addi, adc      C: set if op1 + op2 > 255
sub,  subi          C: set if op1 < op2 (unsigned)
neg, asr            C: set if op1 was odd
sbc                 C: set if op1 + C < op2 (unsigned)
```

```
Setting sreg for cp
if (op1 > op2)   set N else clear N
if (op1 == op2)  set Z else clear Z
if (op1 < op2)   set C else clear C
```

```
PIND:  IO reg 16
DDRD:  IO reg 17
PORTD: IO reg 18
SPL: IO reg 61
SPH: IO reg 62
```

Asm Directive	Example
labels	label_name:
byte	.byte(label_name) 0,1,1,2,3,5,8
string	.string(label_name) "hello world"

ISA Operations	
lo8	ldi r26, lo8(label_name)
hi8	ldi r27, hi8(label_name)

Program layout

Prog memory

```
ldi r31, 91
out 62, r31
ldi r31, 136
out 61, r31      → set sp
rjmp prog_beg    → jump to program start
code for functions
code for functions
code for functions
code for functions
code for functions
program_beg:
code for main prog.
code for main prog.
code for main prog.
code for main prog.
code for main prog.
```

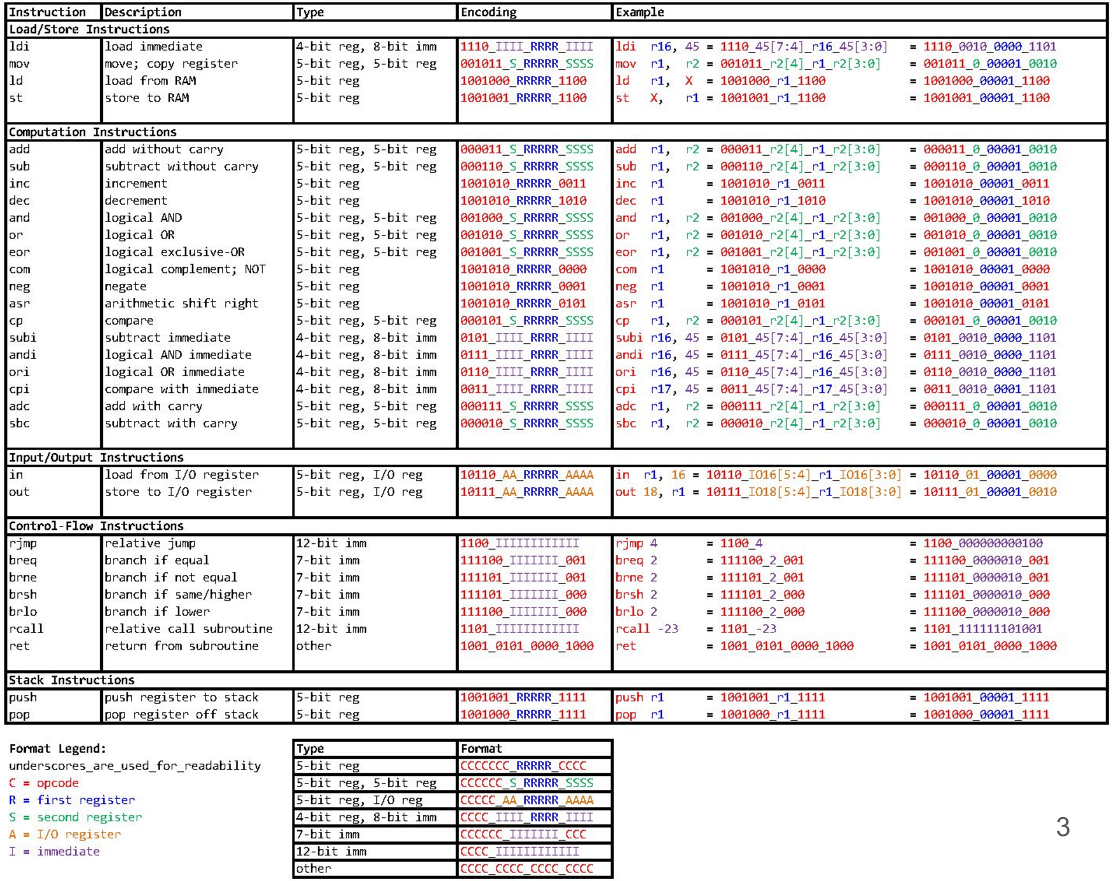
```
if (x < y)
  foo
else
  bar
rest of program

;x in r1,y in r2
cp r1, r2
brlo foo_code
bar line 1
bar line 2
bar line 3
rjmp merge_point
foo_code:
foo line 1
foo line 2
foo line 3
merge_point:
rest of prog.
```

```
if (x == y)
;x in r1,y in r2
cp r1, r2
breq foo_code
```

```
if (x ==10)
;x in r1,y in r2
ldi r31, 10
cp r1, r31
breq foo_code
```

```
if (x >= y)
;x in r1,y in r2
cp r1, r2
brsh foo_code
```

```
write reg 2 to output
ldi r31, 255
out 17, r31
out 18, r2
```



Instruction	Description	Type	Encoding	Example
Load/Store Instructions				
ldi	load immediate	4-bit reg, 8-bit imm	1110_IIII_RRRR_IIII	ldi r16, 45 = 1110_45[7:4]_r16_45[3:0] = 1110_0010_0000_1101
mov	move; copy register	5-bit reg, 5-bit reg	001011_S_RRRRR_SSSS	mov r1, r2 = 001011_r2[4]_r1_r2[3:0] = 001011_0_00001_0010
ld	load from RAM	5-bit reg	1001000_RRRRR_1100	ld r1, X = 1001000_r1_1100 = 1001000_00001_1100
st	store to RAM	5-bit reg	1001001_RRRRR_1100	st X, r1 = 1001001_r1_1100 = 1001001_00001_1100
Computation Instructions				
add	add without carry	5-bit reg, 5-bit reg	000011_S_RRRRR_SSSS	add r1, r2 = 000011_r2[4]_r1_r2[3:0] = 000011_0_00001_0010
sub	subtract without carry	5-bit reg, 5-bit reg	000110_S_RRRRR_SSSS	sub r1, r2 = 000110_r2[4]_r1_r2[3:0] = 000110_0_00001_0010
inc	increment	5-bit reg	1001010_RRRRR_0011	inc r1 = 1001010_r1_0011 = 1001010_00001_0011
dec	decrement	5-bit reg	1001010_RRRRR_1010	dec r1 = 1001010_r1_1010 = 1001010_00001_1010
and	logical AND	5-bit reg, 5-bit reg	001000_S_RRRRR_SSSS	and r1, r2 = 001000_r2[4]_r1_r2[3:0] = 001000_0_00001_0010
or	logical OR	5-bit reg, 5-bit reg	001010_S_RRRRR_SSSS	or r1, r2 = 001010_r2[4]_r1_r2[3:0] = 001010_0_00001_0010
eor	logical exclusive-OR	5-bit reg, 5-bit reg	001001_S_RRRRR_SSSS	eor r1, r2 = 001001_r2[4]_r1_r2[3:0] = 001001_0_00001_0010
com	logical complement; NOT	5-bit reg	1001010_RRRRR_0000	com r1 = 1001010_r1_0000 = 1001010_00001_0000
neg	negate	5-bit reg	1001010_RRRRR_0001	neg r1 = 1001010_r1_0001 = 1001010_00001_0001
asr	arithmetic shift right	5-bit reg	1001010_RRRRR_0101	asr r1 = 1001010_r1_0101 = 1001010_00001_0101
cp	compare	5-bit reg, 5-bit reg	000101_S_RRRRR_SSSS	cp r1, r2 = 000101_r2[4]_r1_r2[3:0] = 000101_0_00001_0010
subi	subtract immediate	4-bit reg, 8-bit imm	0101_IIII_RRRR_IIII	subi r16, 45 = 0101_45[7:4]_r16_45[3:0] = 0101_0010_0000_1101
andi	logical AND immediate	4-bit reg, 8-bit imm	0111_IIII_RRRR_IIII	andi r16, 45 = 0111_45[7:4]_r16_45[3:0] = 0111_0010_0000_1101
ori	logical OR immediate	4-bit reg, 8-bit imm	0110_IIII_RRRR_IIII	ori r16, 45 = 0110_45[7:4]_r16_45[3:0] = 0110_0010_0000_1101
cpi	compare with immediate	4-bit reg, 8-bit imm	0011_IIII_RRRR_IIII	cpi r17, 45 = 0011_45[7:4]_r17_45[3:0] = 0011_0010_0001_1101
adc	add with carry	5-bit reg, 5-bit reg	000111_S_RRRRR_SSSS	adc r1, r2 = 000111_r2[4]_r1_r2[3:0] = 000111_0_00001_0010
sbc	subtract with carry	5-bit reg, 5-bit reg	000010_S_RRRRR_SSSS	sbc r1, r2 = 000010_r2[4]_r1_r2[3:0] = 000010_0_00001_0010
Input/Output Instructions				
in	load from I/O register	5-bit reg, I/O reg	10110_AA_RRRRR_AAAA	in r1, 16 = 10110_IO16[5:4]_r1_IO16[3:0] = 10110_01_00001_0000
out	store to I/O register	5-bit reg, I/O reg	10111_AA_RRRRR_AAAA	out 18, r1 = 10111_IO18[5:4]_r1_IO18[3:0] = 10111_01_00001_0010
Control-Flow Instructions				
rjmp	relative jump	12-bit imm	1100_IIIIIIIIIIII	rjmp 4 = 1100_4 = 1100_000000000100
breq	branch if equal	7-bit imm	111100_IIIIIII_001	breq 2 = 111100_2_001 = 111100_0000010_001
brne	branch if not equal	7-bit imm	111101_IIIIIII_001	brne 2 = 111101_2_001 = 111101_0000010_001
brsh	branch if same/higher	7-bit imm	111101_IIIIIII_000	brsh 2 = 111101_2_000 = 111101_0000010_000
brlo	branch if lower	7-bit imm	111100_IIIIIII_000	brlo 2 = 111100_2_000 = 111100_0000010_000
rcall	relative call subroutine	12-bit imm	1101_IIIIIIIIIIII	rcall -23 = 1101_-23 = 1101_111111101001
ret	return from subroutine	other	1001_0101_0000_1000	ret = 1001_0101_0000_1000 = 1001_0101_0000_1000
Stack Instructions				
push	push register to stack	5-bit reg	1001001_RRRRR_1111	push r1 = 1001001_r1_1111 = 1001001_00001_1111
pop	pop register off stack	5-bit reg	1001000_RRRRR_1111	pop r1 = 1001000_r1_1111 = 1001000_00001_1111

Format Legend:

underscores_are_used_for_readability

C = opcode

R = first register

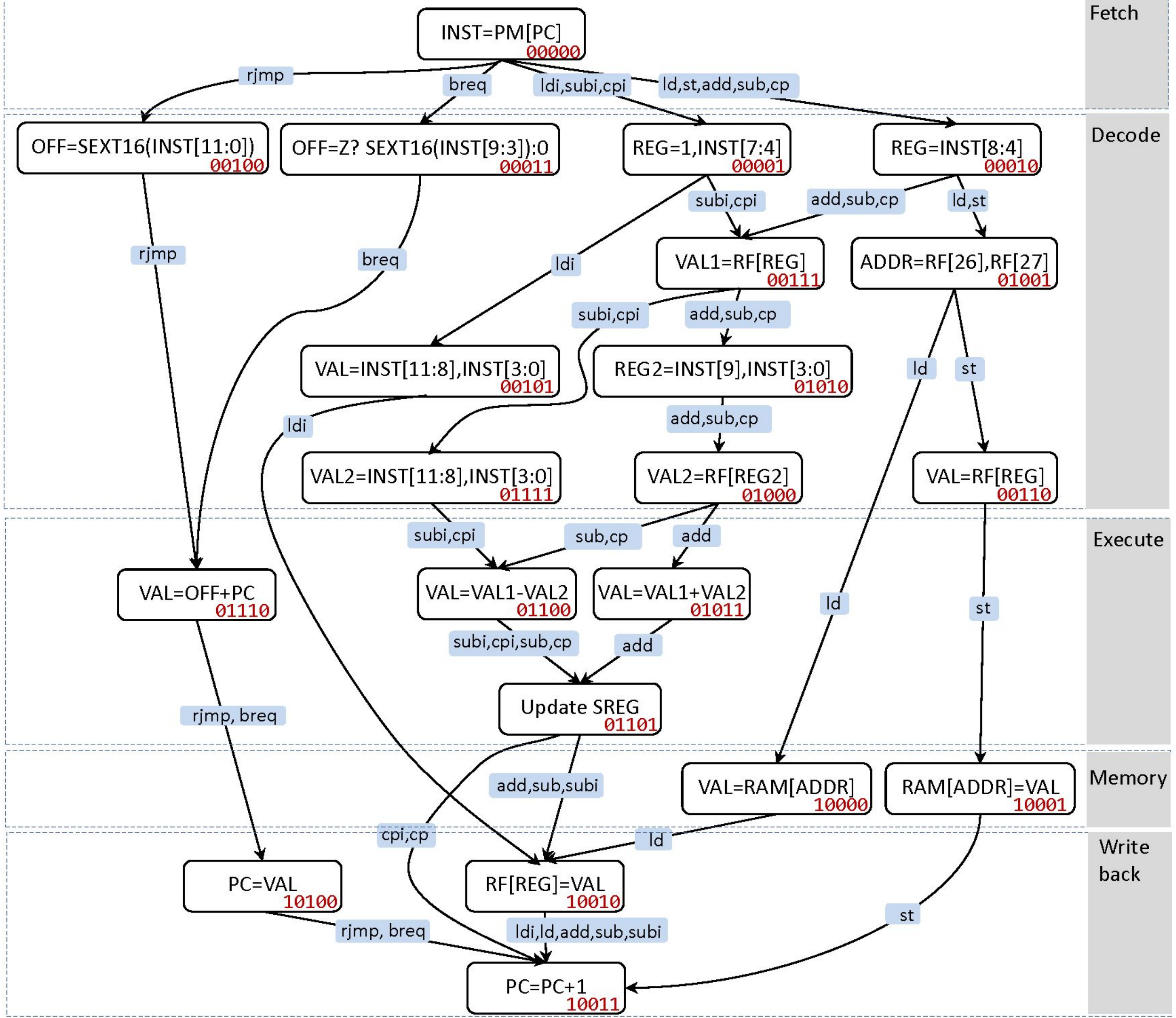
S = second register

A = I/O register

I = immediate

Type	Format
5-bit reg	CCCCCCC_RRRRR_CCCC
5-bit reg, 5-bit reg	CCCCCC_S_RRRRR_SSSS
5-bit reg, I/O reg	CCCCC_AA_RRRRR_AAAA
4-bit reg, 8-bit imm	CCCC_IIII_RRRR_IIII
7-bit imm	CCCCCC_IIIIIII_CCC
12-bit imm	CCCC_IIIIIIIIIIII
other	CCCC_CCCC_CCCC_CCCC

Cheatsheet 3: Chapter 7, State Machine



Fetch

INST=PM[PC] 00000

rjmp

breq

ldi,subi,cpi

ld,st,add,sub,cp

Decode

OFF=SEXT16(INST[11:0]) 00100

OFF=Z? SEXT16(INST[9:3]):0 00011

REG=1,INST[7:4] 00001

REG=INST[8:4] 00010

subi,cpi

add,sub,cp

ld,st

rjmp

breq

ldi

VAL1=RF[REG] 00111

ADDR=RF[26],RF[27] 01001

subi,cpi

add,sub,cp

VAL=INST[11:8],INST[3:0] 00101

REG2=INST[9],INST[3:0] 01010

ld

st

ldi

add,sub,cp

VAL2=INST[11:8],INST[3:0] 01111

VAL2=RF[REG2] 01000

VAL=RF[REG] 00110

Execute

subi,cpi

sub,cp

add

VAL=OFF+PC 01110

VAL=VAL1-VAL2 01100

VAL=VAL1+VAL2 01011

ld

st

subi,cpi,sub,cp

add

Update SREG 01101

rjmp, breq

Memory

add,sub,subi

VAL=RAM[ADDR] 10000

RAM[ADDR]=VAL 10001

Write back

cpi,cp

ld

PC=VAL 10100

RF[REG]=VAL 10010

st

rjmp, breq

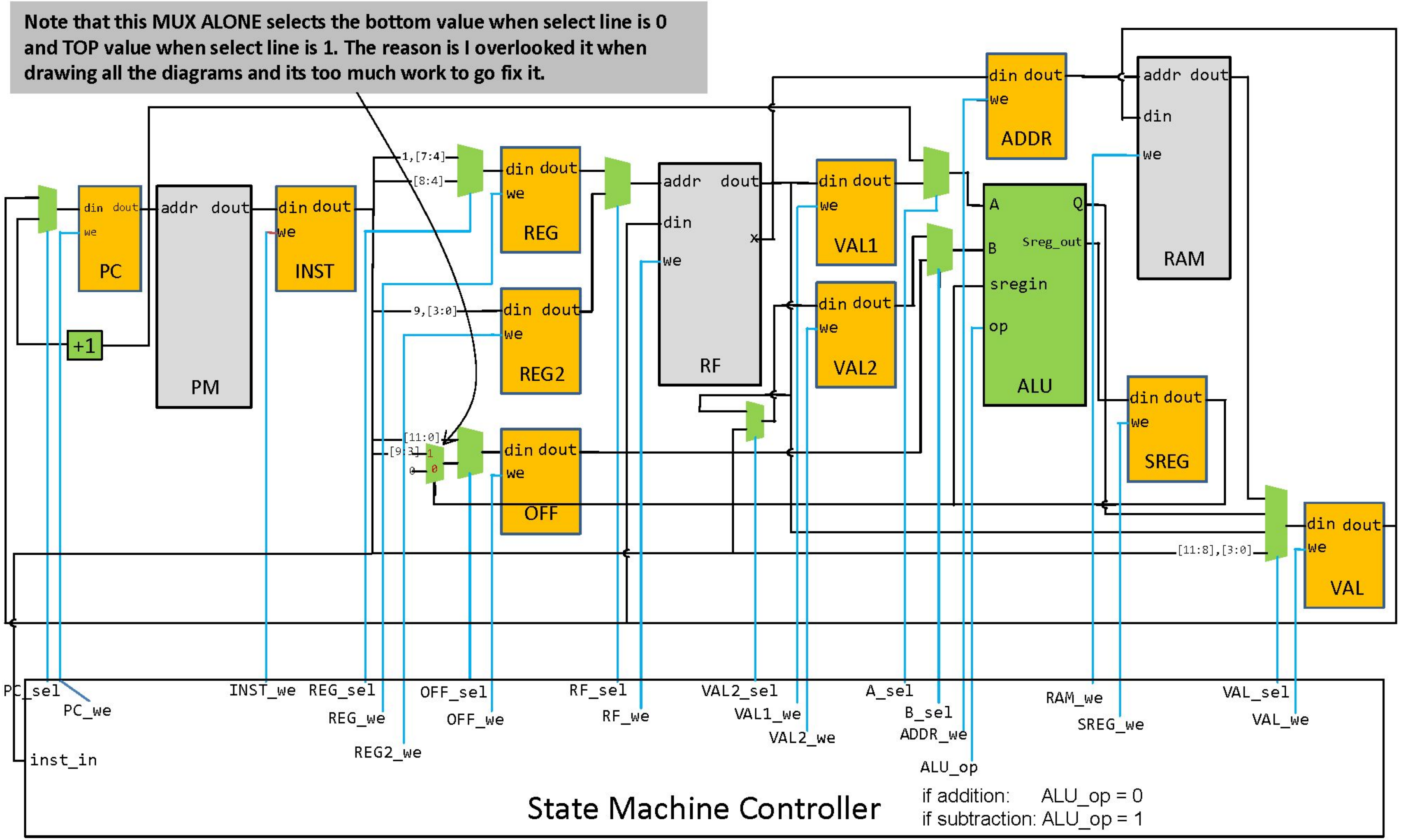
ldi,ld,add,sub,subi

PC=PC+1 10011

SEXT16: sign extend to 16 bits

Cheatsheet 4: Chapter 7, Microarchitecture

Note that this MUX ALONE selects the bottom value when select line is 0 and TOP value when select line is 1. The reason is I overlooked it when drawing all the diagrams and its too much work to go fix it.



if addition: ALU_op = 0
if subtraction: ALU_op = 1

Exam 3 Topics - Chapter 6

Encoding Principles - section6.2, lecture18_slide18

Encoding - section6.3.2, HW6Q13; lecture18

Decoding - section7.2.1.1, section7.3.1through7.3.7, HW6Q14; lecture18

color legend: book, homework, lecture

Exam 3 Topics - Chapter 7

Microarchitecture - section7.2, lecture19slide3-5

State Machines - section7.2through7.2.1.1, HW7Q1, HW7Q2, HW7Q11, HW7Q12, lecture19slide8-15, lecture22slide4-11

Microarchitecture execution stages - lecture19_slide10, section7.2through7.2.1.1, HW7Q5

Circuit components - section7.2.2, HW7Q12, lecture20

- wires and buses (bus = collection of wires) - section7.2.2, HW7Q7, HW7Q8
- memories - section7.2.2
- arithmetic logic unit (ALU) - section7.2.2, HW7Q8
- multiplexer (mux) - section7.2.2, HW7Q7, HW7Q8, lecture22slide12-19
- program memory (PM) - section7.4.1
- random access memory (RAM) - section7.4.1
- register file (RF) - section7.4.1
- program counter (PC), status register (SREG), and auxiliary registers - section7.4.1, HW7Q4, HW7Q6, lecture19slide16, lecture23slide5, lectures25-27

color legend: book, homework, lecture

Exam 3 Topics - Chapter 7 (continued)

Instruction Implementation

- ldi - section7.3.1and7.4.2, HW7Q9, lecture19slide17-19, lecture20slide13-20
- ld - section7.3.2and7.4.3, HW7Q9, lecture19slide20-22, lecture23slide6-20
- st, add, sub, cp, subi, cpi, rjmp, breq - section7.3.3through7.4.4
- even more - lecture26slide10-19
  - From lecture26slide19: implementing instructions will be on the exam
    - exam won’t ask you to implement any instruction that involves I/O registers

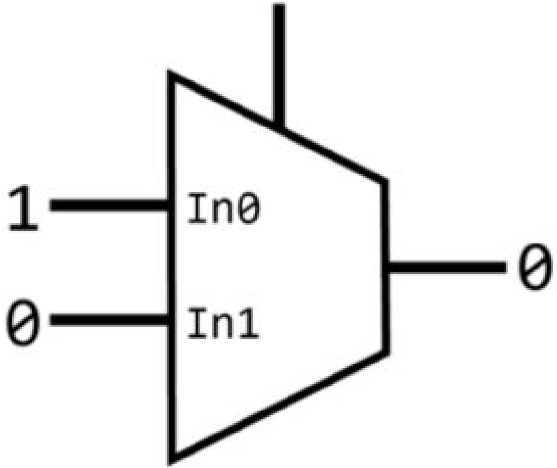
State Transition - section7.2.1, HW7Q11, lecture22slide21-23, lecture24slide5-21, lectures25-27

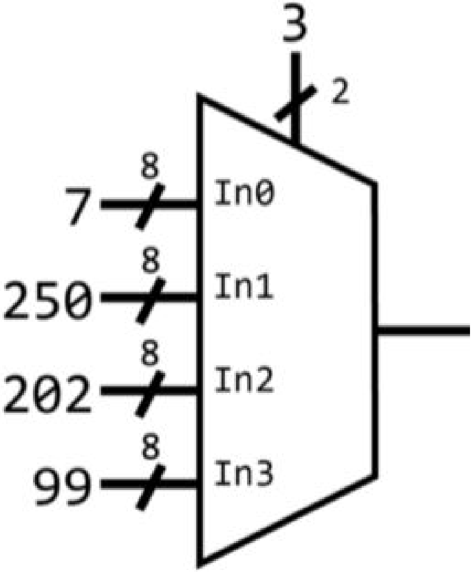
Tracing Instruction Execution in Hardware - chapter7, HW7Q11, HW7Q12, lectures25-27

color legend: book, homework, lecture

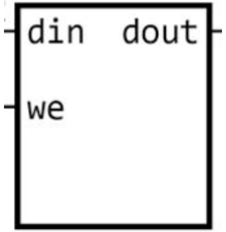
MUX: Notation & Selection

Use the selector to indicate which input should be used as output.





AUX registers



- Auxiliary registers have three ports: **din**, **dout**, and **we**
- When **we** is low (0) **dout** ignores **din**, and continues to be set to what **din** was before **we** went low.
- When **we** is high (1) **dout = din**

WE	0	1	0
DIN	1	1	0
DOUT			

Binary -> ISA

```
1110000000001111 -> ldi ?, ?
```

What do I need to look for?

Binary -> ISA

1. Find the instruction format: `CCCC_IIII_RRRR_IIII`
2. Divide binary: `1110_0000_0000_1111`
3. Find the opcode: `1110`
4. Calculate:
   a. `0000_1111` -> 15
   b. `0000` -> 0 -> r16
5. Instruction: r16, 15

ISA-> Binary

add r17, r19

How do we convert this to binary?

ISA -> Binary

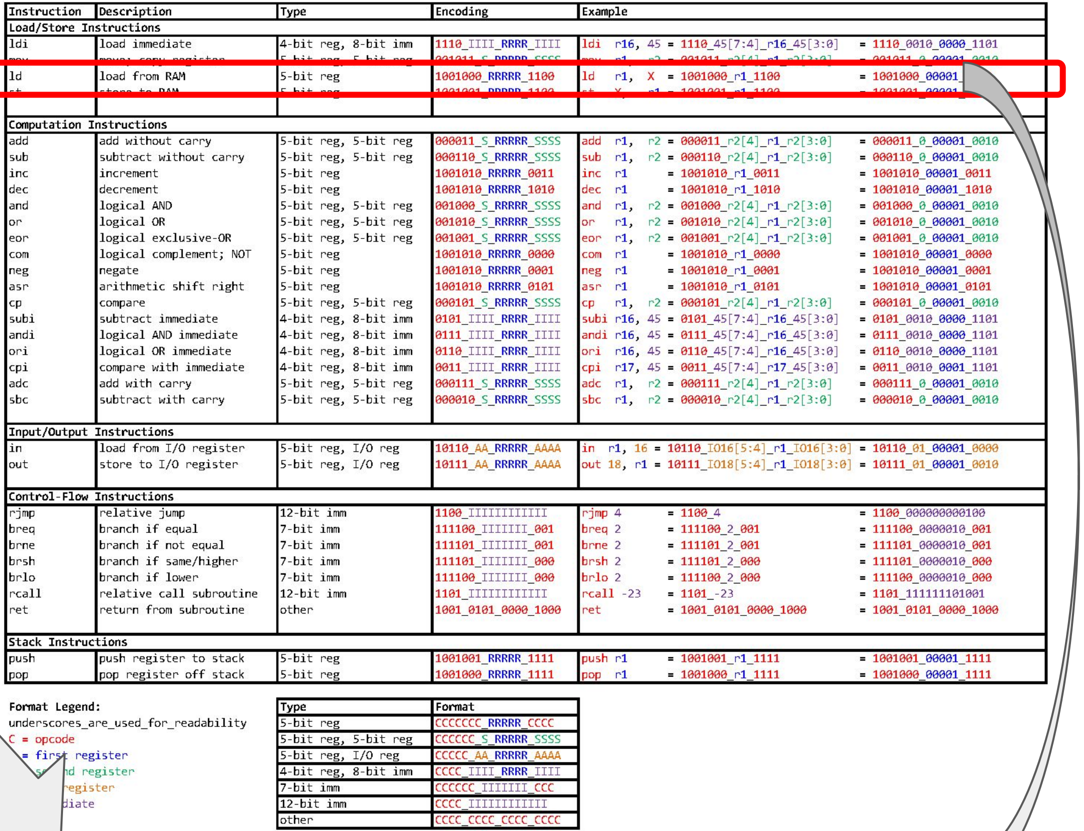
1. Look up the format: `CCCCCC_S_RRRRR_SSSS`
2. Look up the opcode: `000011`
3. Convert C, R, S
    a. C = 00011
    b. R = r17 = 10001
    c. S = r19 = 10011
4. Insert: `000011_1_10001_0011 -> 0000_1111_0001_0011`

Tracing

```
; assume this is the entire program,
;   so r13, r26 and r27 is initially 0
ld r27, X       ; demonstrates accessing RAM
cp r13, r27     ; demonstrates updating SREG
rjmp -23        ; demonstrates negative immediates and SEXT
; this is a bad program, so
```

Tracing ld Encoding

```
ld r27, X
27 = 16 + 8 + 2 + 1
   = 0b11011
```



Instruction	Description	Type	Encoding	Example
Load/Store Instructions				
ldi	load immediate	4-bit reg, 8-bit imm	1110_IIII_RRRR_IIII	ldi r16, 45 = 1110_45[7:4]_r16_45[3:0] = 1110_0010_0000_1101
mov	move; copy register	5-bit reg, 5-bit reg	001011_S_RRRRR_SSSS	mov r1, r2 = 001011_r2[4]_r1_r2[3:0] = 001011_0_00001_0010
ld	load from RAM	5-bit reg	1001000_RRRRR_1100	ld r1, X = 1001000_r1_1100 = 1001000_00001_
st	store to RAM	5-bit reg	1001001_RRRRR_1100	st X, r1 = 1001001_r1_1100 = 1001001_00001_
Computation Instructions				
add	add without carry	5-bit reg, 5-bit reg	000011_S_RRRRR_SSSS	add r1, r2 = 000011_r2[4]_r1_r2[3:0] = 000011_0_00001_0010
sub	subtract without carry	5-bit reg, 5-bit reg	000110_S_RRRRR_SSSS	sub r1, r2 = 000110_r2[4]_r1_r2[3:0] = 000110_0_00001_0010
inc	increment	5-bit reg	1001010_RRRRR_0011	inc r1 = 1001010_r1_0011 = 1001010_00001_0011
dec	decrement	5-bit reg	1001010_RRRRR_1010	dec r1 = 1001010_r1_1010 = 1001010_00001_1010
and	logical AND	5-bit reg, 5-bit reg	001000_S_RRRRR_SSSS	and r1, r2 = 001000_r2[4]_r1_r2[3:0] = 001000_0_00001_0010
or	logical OR	5-bit reg, 5-bit reg	001010_S_RRRRR_SSSS	or r1, r2 = 001010_r2[4]_r1_r2[3:0] = 001010_0_00001_0010
eor	logical exclusive-OR	5-bit reg, 5-bit reg	001001_S_RRRRR_SSSS	eor r1, r2 = 001001_r2[4]_r1_r2[3:0] = 001001_0_00001_0010
com	logical complement; NOT	5-bit reg	1001010_RRRRR_0000	com r1 = 1001010_r1_0000 = 1001010_00001_0000
neg	negate	5-bit reg	1001010_RRRRR_0001	neg r1 = 1001010_r1_0001 = 1001010_00001_0001
asr	arithmetic shift right	5-bit reg	1001010_RRRRR_0101	asr r1 = 1001010_r1_0101 = 1001010_00001_0101
cp	compare	5-bit reg, 5-bit reg	000101_S_RRRRR_SSSS	cp r1, r2 = 000101_r2[4]_r1_r2[3:0] = 000101_0_00001_0010
subi	subtract immediate	4-bit reg, 8-bit imm	0101_IIII_RRRR_IIII	subi r16, 45 = 0101_45[7:4]_r16_45[3:0] = 0101_0010_0000_1101
andi	logical AND immediate	4-bit reg, 8-bit imm	0111_IIII_RRRR_IIII	andi r16, 45 = 0111_45[7:4]_r16_45[3:0] = 0111_0010_0000_1101
ori	logical OR immediate	4-bit reg, 8-bit imm	0110_IIII_RRRR_IIII	ori r16, 45 = 0110_45[7:4]_r16_45[3:0] = 0110_0010_0000_1101
cpi	compare with immediate	4-bit reg, 8-bit imm	0011_IIII_RRRR_IIII	cpi r17, 45 = 0011_45[7:4]_r17_45[3:0] = 0011_0010_0001_1101
adc	add with carry	5-bit reg, 5-bit reg	000111_S_RRRRR_SSSS	adc r1, r2 = 000111_r2[4]_r1_r2[3:0] = 000111_0_00001_0010
sbc	subtract with carry	5-bit reg, 5-bit reg	000010_S_RRRRR_SSSS	sbc r1, r2 = 000010_r2[4]_r1_r2[3:0] = 000010_0_00001_0010
Input/Output Instructions				
in	load from I/O register	5-bit reg, I/O reg	10110_AA_RRRRR_AAAA	in r1, 16 = 10110_IO16[5:4]_r1_IO16[3:0] = 10110_01_00001_0000
out	store to I/O register	5-bit reg, I/O reg	10111_AA_RRRRR_AAAA	out 18, r1 = 10111_IO18[5:4]_r1_IO18[3:0] = 10111_01_00001_0010
Control-Flow Instructions				
rjmp	relative jump	12-bit imm	1100_IIIIIIIIIIII	rjmp 4 = 1100_4 = 1100_000000000100
breq	branch if equal	7-bit imm	111100_IIIIIII_001	breq 2 = 111100_2_001 = 111100_0000010_001
brne	branch if not equal	7-bit imm	111101_IIIIIII_001	brne 2 = 111101_2_001 = 111101_0000010_001
brsh	branch if same/higher	7-bit imm	111101_IIIIIII_000	brsh 2 = 111101_2_000 = 111101_0000010_000
brlo	branch if lower	7-bit imm	111100_IIIIIII_000	brlo 2 = 111100_2_000 = 111100_0000010_000
rcall	relative call subroutine	12-bit imm	1101_IIIIIIIIIIII	rcall -23 = 1101_-23 = 1101_111111101001
ret	return from subroutine	other	1001_0101_0000_1000	ret = 1001_0101_0000_1000 = 1001_0101_0000_1000
Stack Instructions				
push	push register to stack	5-bit reg	1001001_RRRRR_1111	push r1 = 1001001_r1_1111 = 1001001_00001_1111
pop	pop register off stack	5-bit reg	1001000_RRRRR_1111	pop r1 = 1001000_r1_1111 = 1001000_00001_1111

Format Legend:
underscores_are_used_for_readability
C = opcode
= first register
s nd register
egister
diate

Type	Format
5-bit reg	CCCCCCC_RRRRR_CCCC
5-bit reg, 5-bit reg	CCCCCC_S_RRRRR_SSSS
5-bit reg, I/O reg	CCCCC_AA_RRRRR_AAAA
4-bit reg, 8-bit imm	CCCC_IIII_RRRR_IIII
7-bit imm	CCCCCC_IIIIIII_CCC
12-bit imm	CCCC_IIIIIIIIIIII
other	CCCC_CCCC_CCCC_CCCC

Instruction	Description	Type	Encoding	Example
ld	load from RAM	5-bit reg	1001000_RRRRR_1100	ld r1, X = 1001000_r1_1100 = 1001000_00001_1100

```
INST = 0b100100_11011_1100 = 0b1001_0001_1011_1100 = 0x91BC = 37308
```

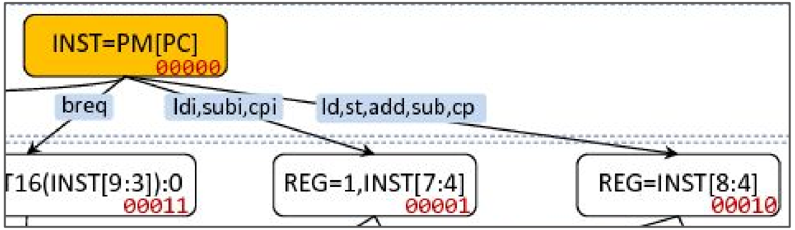
Tracing ld Cycle 1

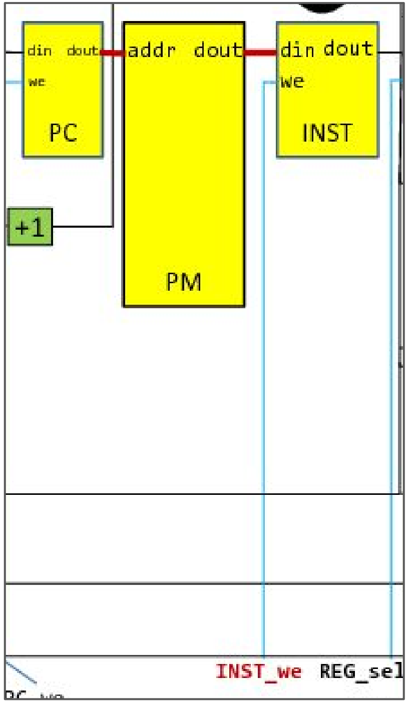
INST = 0b100100_11011_1100 = 0b1001_0001_1011_1100 = 0x91BC = 37308

Cycle	State	Changed registers/values and control signals

1	00000	INST = 0x91BC,
		INST_we = 1,
		other_ctrls = 0

2		
3		
4		
5		
6		





Tracing ld Cycle 2

Cycle	State	Changed registers/values and control signals

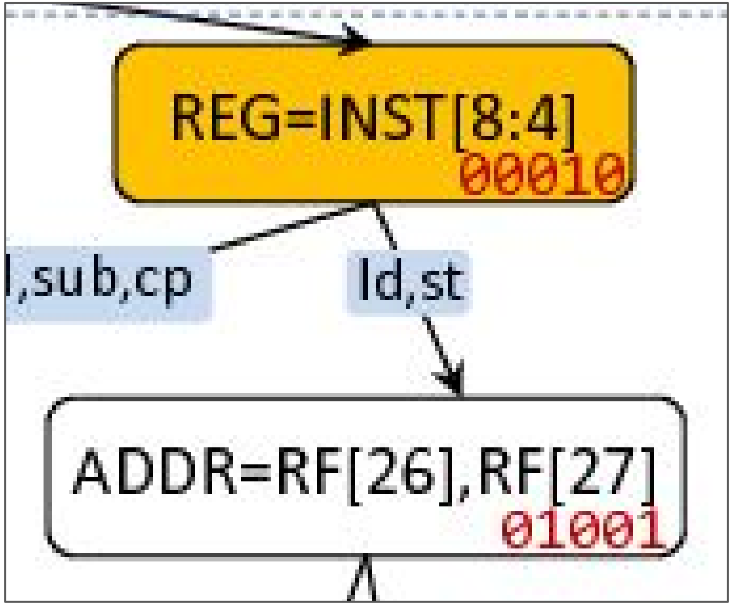
1	00000	INST = 0x91BC,
		INST_we = 1,
		other_ctrls = 0

2	00010	REG = 27, REG_we = 1,
		REG_sel = 1,
		other_ctrls = 0

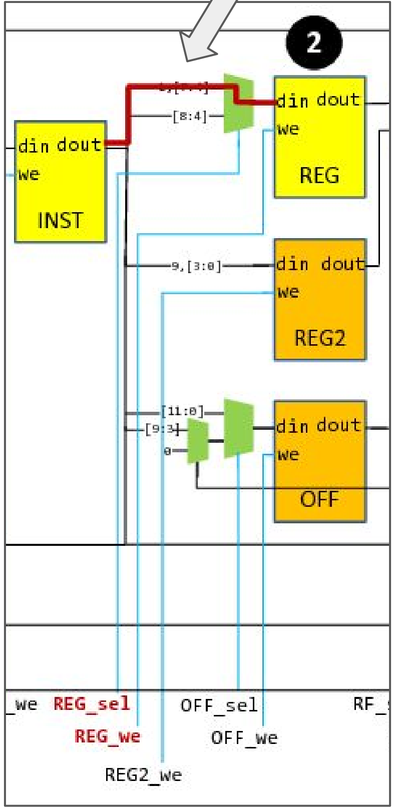
3		
4		
5		
6		

INST[15] INST[8] INST[4] INST[0]

```
INST = INST[15:0]   = 0b1001_0001_1011_1100
INST[8:4]           =        0b1_1011
REG = INST[8:4]     =        0b1_1011 = 27
```



simulator error



Tracing ld Cycle 3

```
; assume this is the entire program,
;   so r13, r26 and r27 is initially 0
RF[26] = 0x00, RF[27] = 0x00
ADDR = RF[26],RF[27] = 0x00,0x00 = 0x0000 = 0
```

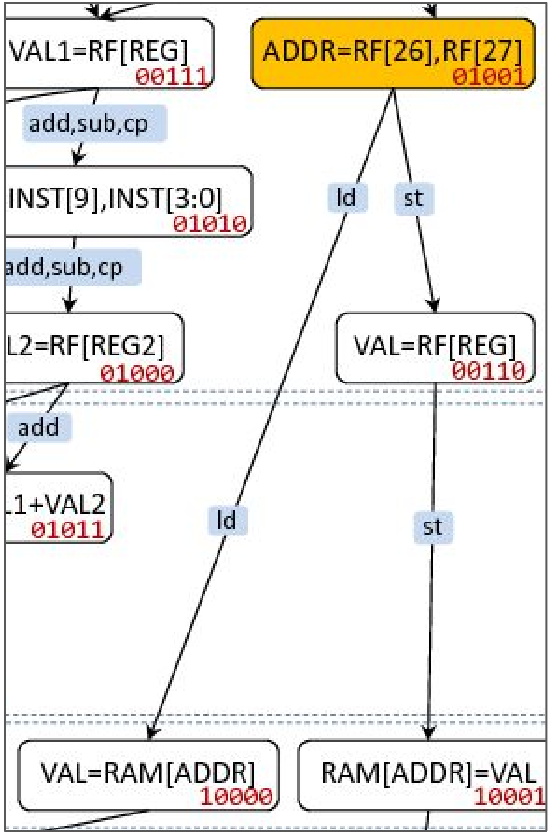
Cycle	State	Changed registers/values and control signals

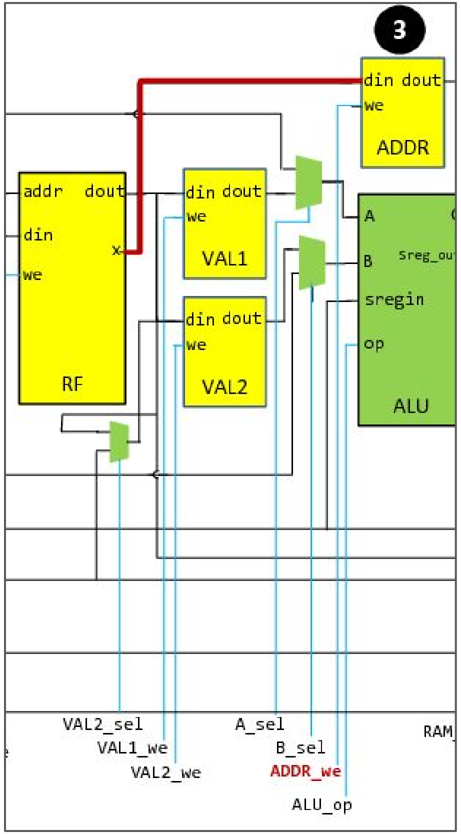
1	00000	INST = 0x91BC,
		INST_we = 1,
		other_ctrls = 0

2	00010	REG = 27, REG_we = 1,
		REG_sel = 1,
		other_ctrls = 0

3	01001	ADDR = 0, ADDR_we = 1,
		other_ctrls = 0

4		
5		
6		





Tracing ld Cycle 4

```
; assume this is the entire program,
;    so no assembler directive and all RAM is 0
ADDR = 0
VAL = RAM[ADDR] = RAM[0] = 0
```

Cycle	State	Changed registers/values and control signals

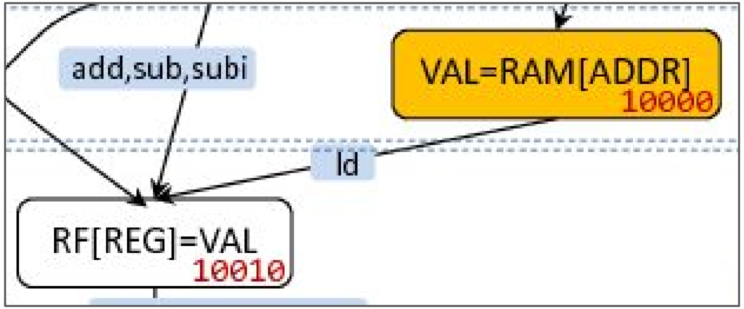
1	00000	INST = 0x91BC,
		INST_we = 1,
		other_ctrls = 0

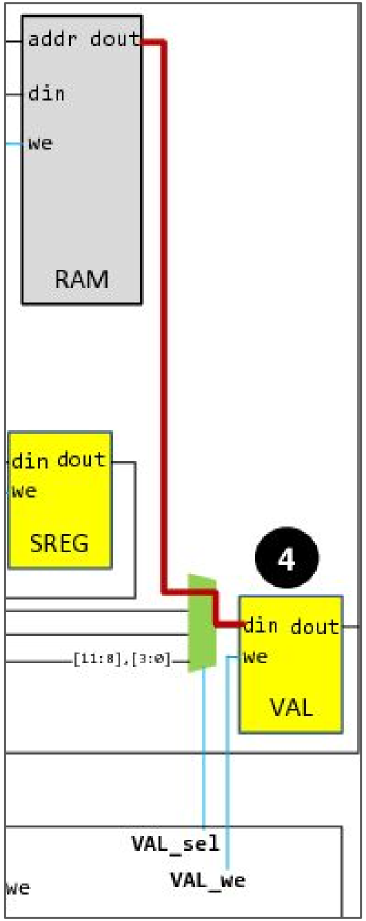
2	00010	REG = 27, REG_we = 1,
		REG_sel = 1,
		other_ctrls = 0

3	01001	ADDR = 0, ADDR_we = 1,
		other_ctrls = 0

4	10000	VAL = 0, VAL_we = 1,
		VAL_sel = 0,
		other_ctrls = 0

5		
6		





Tracing ld Cycle 5

```
VAL = 0, REG = 27
RF[REG] = VAL
RF[27] = 0
```

Cycle	State	Changed registers/values and control signals

1	00000	INST = 0x91BC,
		INST_we = 1,
		other_ctrls = 0

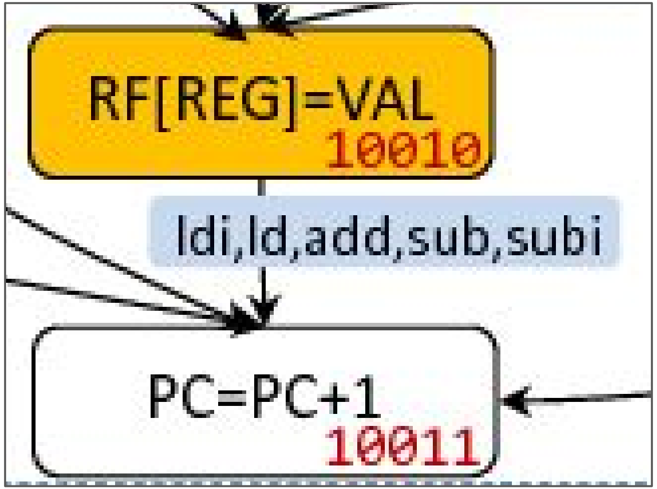
2	00010	REG = 27, REG_we = 1,
		REG_sel = 1,
		other_ctrls = 0

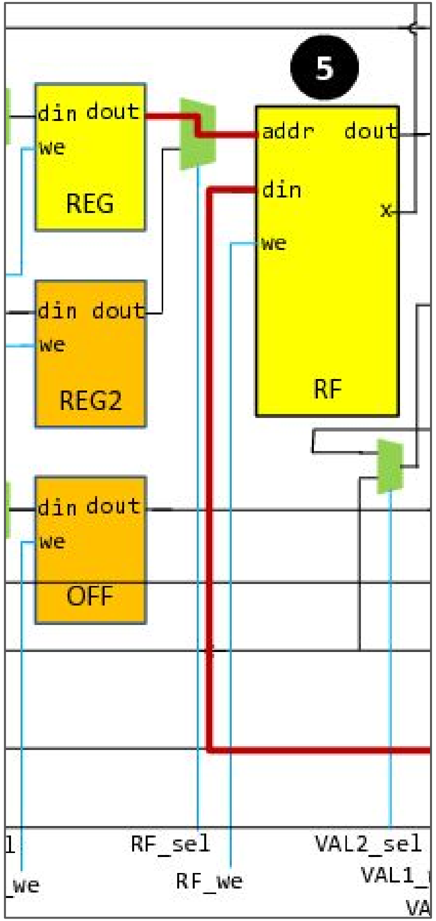
3	01001	ADDR = 0, ADDR_we = 1,
		other_ctrls = 0

4	10000	VAL = 0, VAL_we = 1,
		VAL_sel = 0,
		other_ctrls = 0

5	10010	r27 = 0, RF_we = 1,
		RF_sel = 0,
		other_ctrls = 0

6		





Tracing ld Cycle 6

PC = 0

PC = PC + 1 = 0 + 1 = 1

Cycle	State	Changed registers/values and control signals

1	00000	INST = 0x91BC,
		INST_we = 1,
		other_ctrls = 0

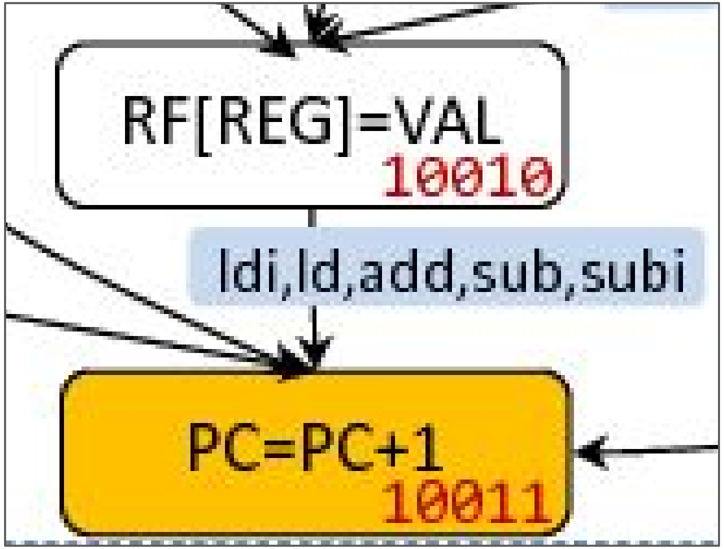
2	00010	REG = 27, REG_we = 1,
		REG_sel = 1,
		other_ctrls = 0

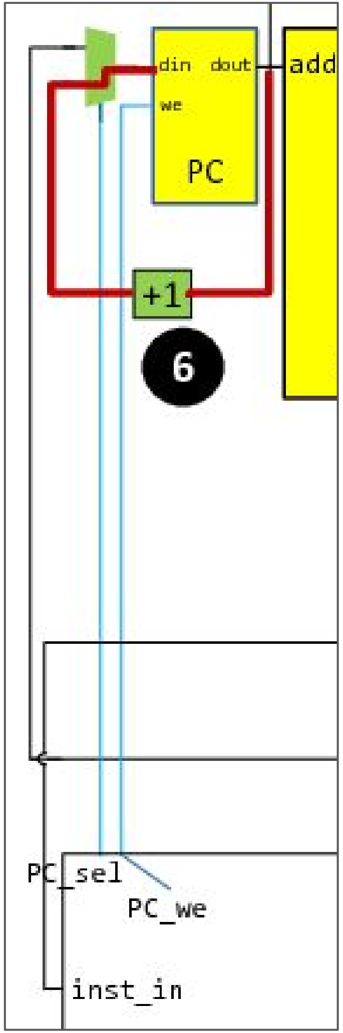
3	01001	ADDR = 0, ADDR_we = 1,
		other_ctrls = 0

4	10000	VAL = 0, VAL_we = 1,
		VAL_sel = 0,
		other_ctrls = 0

5	10010	r27 = 0, RF_we = 1,
		RF_sel = 0,
		other_ctrls = 0

6	10011	PC = 1, PC_we = 1
		PC_sel = 1,
		other_ctrls = 0



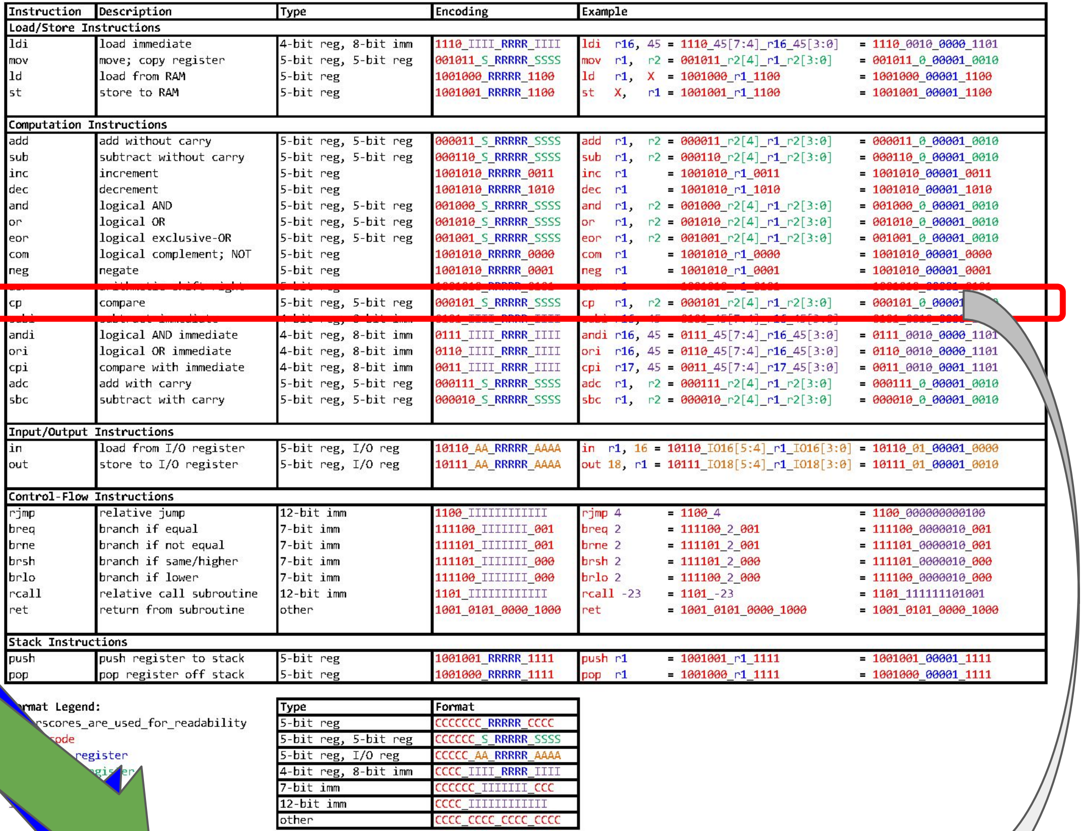


Tracing cp Encoding

cp r13, r27

13 = 8 + 4 + 1
   = 0b01101

27 = 16 + 8 + 2 + 1
   = 0b11011



Instruction	Description	Type	Encoding	Example
Load/Store Instructions				
ldi	load immediate	4-bit reg, 8-bit imm	1110_IIII_RRRR_IIII	ldi r16, 45 = 1110_45[7:4]_r16_45[3:0] = 1110_0010_0000_1101
mov	move; copy register	5-bit reg, 5-bit reg	001011_S_RRRRR_SSSS	mov r1, r2 = 001011_r2[4]_r1_r2[3:0] = 001011_0_00001_0010
ld	load from RAM	5-bit reg	1001000_RRRRR_1100	ld r1, X = 1001000_r1_1100 = 1001000_00001_1100
st	store to RAM	5-bit reg	1001001_RRRRR_1100	st X, r1 = 1001001_r1_1100 = 1001001_00001_1100
Computation Instructions				
add	add without carry	5-bit reg, 5-bit reg	000011_S_RRRRR_SSSS	add r1, r2 = 000011_r2[4]_r1_r2[3:0] = 000011_0_00001_0010
sub	subtract without carry	5-bit reg, 5-bit reg	000110_S_RRRRR_SSSS	sub r1, r2 = 000110_r2[4]_r1_r2[3:0] = 000110_0_00001_0010
inc	increment	5-bit reg	1001010_RRRRR_0011	inc r1 = 1001010_r1_0011 = 1001010_00001_0011
dec	decrement	5-bit reg	1001010_RRRRR_1010	dec r1 = 1001010_r1_1010 = 1001010_00001_1010
and	logical AND	5-bit reg, 5-bit reg	001000_S_RRRRR_SSSS	and r1, r2 = 001000_r2[4]_r1_r2[3:0] = 001000_0_00001_0010
or	logical OR	5-bit reg, 5-bit reg	001010_S_RRRRR_SSSS	or r1, r2 = 001010_r2[4]_r1_r2[3:0] = 001010_0_00001_0010
eor	logical exclusive-OR	5-bit reg, 5-bit reg	001001_S_RRRRR_SSSS	eor r1, r2 = 001001_r2[4]_r1_r2[3:0] = 001001_0_00001_0010
com	logical complement; NOT	5-bit reg	1001010_RRRRR_0000	com r1 = 1001010_r1_0000 = 1001010_00001_0000
neg	negate	5-bit reg	1001010_RRRRR_0001	neg r1 = 1001010_r1_0001 = 1001010_00001_0001
asr	arithmetic shift right	5-bit reg	1001010_RRRRR_0101	asr r1 = 1001010_r1_0101 = 1001010_00001_0101
cp	compare	5-bit reg, 5-bit reg	000101_S_RRRRR_SSSS	cp r1, r2 = 000101_r2[4]_r1_r2[3:0] = 000101_0_00001_0010
subi	subtract immediate	4-bit reg, 8-bit imm	0101_IIII_RRRR_IIII	subi r16, 45 = 0101_45[7:4]_r16_45[3:0] = 0101_0010_0000_1101
andi	logical AND immediate	4-bit reg, 8-bit imm	0111_IIII_RRRR_IIII	andi r16, 45 = 0111_45[7:4]_r16_45[3:0] = 0111_0010_0000_1101
ori	logical OR immediate	4-bit reg, 8-bit imm	0110_IIII_RRRR_IIII	ori r16, 45 = 0110_45[7:4]_r16_45[3:0] = 0110_0010_0000_1101
cpi	compare with immediate	4-bit reg, 8-bit imm	0011_IIII_RRRR_IIII	cpi r17, 45 = 0011_45[7:4]_r17_45[3:0] = 0011_0010_0001_1101
adc	add with carry	5-bit reg, 5-bit reg	000111_S_RRRRR_SSSS	adc r1, r2 = 000111_r2[4]_r1_r2[3:0] = 000111_0_00001_0010
sbc	subtract with carry	5-bit reg, 5-bit reg	000010_S_RRRRR_SSSS	sbc r1, r2 = 000010_r2[4]_r1_r2[3:0] = 000010_0_00001_0010
Input/Output Instructions				
in	load from I/O register	5-bit reg, I/O reg	10110_AA_RRRRR_AAAA	in r1, 16 = 10110_IO16[5:4]_r1_IO16[3:0] = 10110_01_00001_0000
out	store to I/O register	5-bit reg, I/O reg	10111_AA_RRRRR_AAAA	out 18, r1 = 10111_IO18[5:4]_r1_IO18[3:0] = 10111_01_00001_0010
Control-Flow Instructions				
rjmp	relative jump	12-bit imm	1100_IIIIIIIIIIII	rjmp 4 = 1100_4 = 1100_000000000100
breq	branch if equal	7-bit imm	111100_IIIIIII_001	breq 2 = 111100_2_001 = 111100_0000010_001
brne	branch if not equal	7-bit imm	111101_IIIIIII_001	brne 2 = 111101_2_001 = 111101_0000010_001
brsh	branch if same/higher	7-bit imm	111101_IIIIIII_000	brsh 2 = 111101_2_000 = 111101_0000010_000
brlo	branch if lower	7-bit imm	111100_IIIIIII_000	brlo 2 = 111100_2_000 = 111100_0000010_000
rcall	relative call subroutine	12-bit imm	1101_IIIIIIIIIIII	rcall -23 = 1101_-23 = 1101_111111101001
ret	return from subroutine	other	1001_0101_0000_1000	ret = 1001_0101_0000_1000 = 1001_0101_0000_1000
Stack Instructions				
push	push register to stack	5-bit reg	1001001_RRRRR_1111	push r1 = 1001001_r1_1111 = 1001001_00001_1111
pop	pop register off stack	5-bit reg	1001000_RRRRR_1111	pop r1 = 1001000_r1_1111 = 1001000_00001_1111

rmat Legend:
rscores_are_used_for_readability
code
register
gister

Type	Format
5-bit reg	CCCCCCC_RRRRR_CCCC
5-bit reg, 5-bit reg	CCCCCC_S_RRRRR_SSSS
5-bit reg, I/O reg	CCCCC_AA_RRRRR_AAAA
4-bit reg, 8-bit imm	CCCC_IIII_RRRR_IIII
7-bit imm	CCCCCC_IIIIIII_CCC
12-bit imm	CCCC_IIIIIIIIIIII
other	CCCC_CCCC_CCCC_CCCC

Instruction	Description	Type	Encodin	Example
cp	compare	5-bit reg, 5-bit reg	000101_S_RRRRR_SSSS	cp r1, r2 = 000101_r2[4]_r1_r2[3:0] = 000101_0_00001_0010

INST = 0b000101_1_01101_1011 = 0b0001_0110_1101_1011 = 0x16DB = 5851

Tracing cp Cycle 1

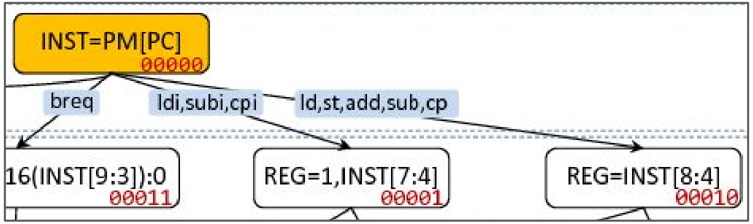
INST = 0b0001_0110_1101_1011

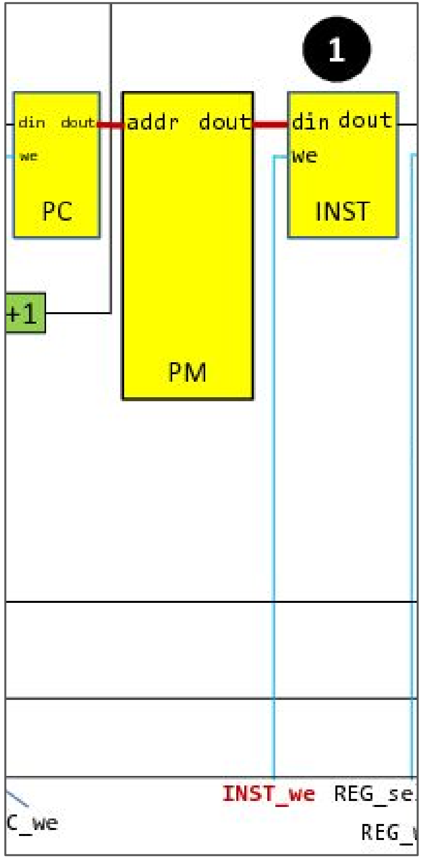
= 0x16DB

Cycle	State	Changed registers/values and control signals

1	00000	INST = 0x16DB,
		INST_we = 1,
		other_ctrls = 0

2		
3		
4		
5		
6		





Tracing cp Cycle 2

Cycle	State	Changed registers/values and control signals

1	00000	INST = 0x16DB,
		INST_we = 1,
		other_ctrls = 0

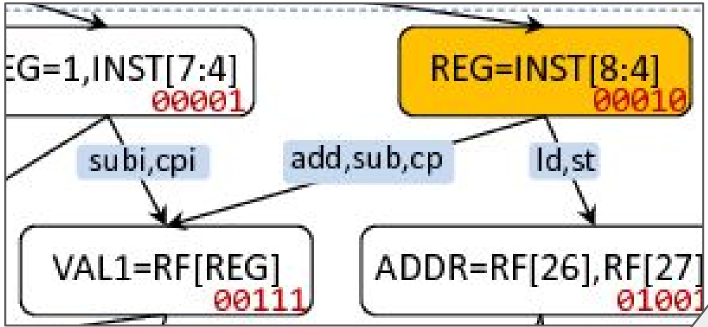
2	00010	REG = 13, REG_we = 1,
		REG_sel = 1,
		other_ctrls = 0

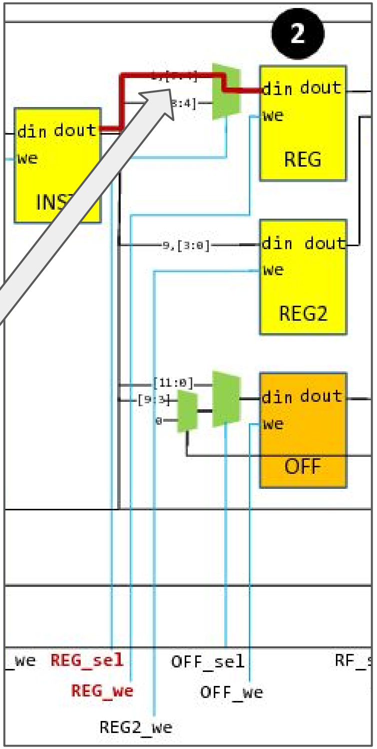
3		
4		
5		
6		

INST[15] INST[8] INST[4] INST[0]

INST = INST[15:0] = 0b0001_0110_1101_1011

REG = INST[8:4] = 0b0_1101 = 13





simulator error

Tracing cp Cycle 3

Cycle	State	Changed registers/values and control signals

1	00000	INST = 0x16DB,
		INST_we = 1,
		other_ctrls = 0

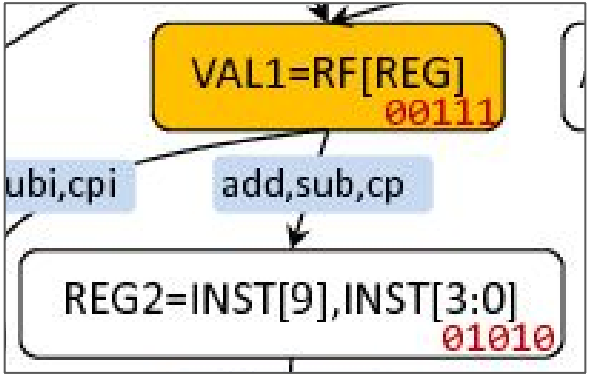
2	00010	REG = 13, REG_we = 1,
		REG_sel = 1,
		other_ctrls = 0

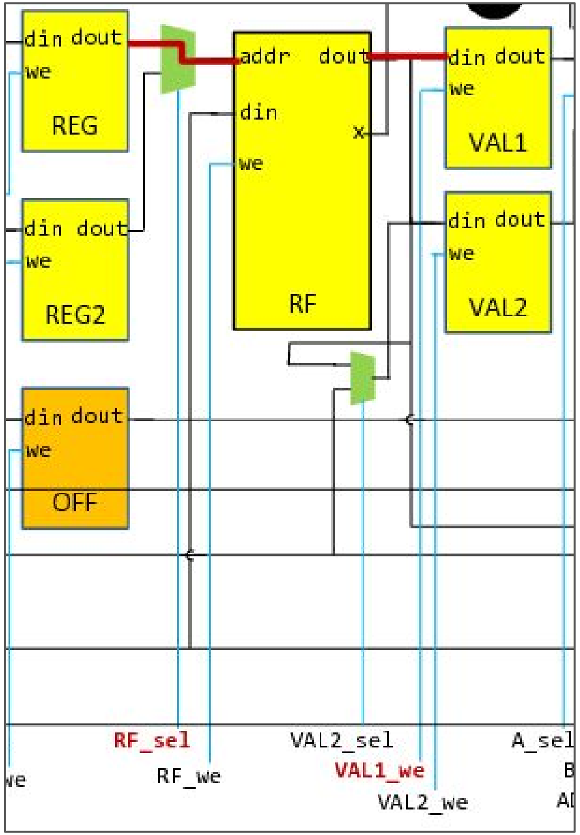
3	00111	VAL1 = 0, VAL1_we = 1,
		RF_sel = 0,
		other_ctrls = 0

4		
5		
6		

```
; assume this is the entire program,
;   so r13, r26 and r27 is initially 0
```

```
REG = 13
VAL1 = RF[REG]
     = RF[13] = 0
```





Tracing cp Cycle 4

Cycle	State	Changed registers/values and control signals

1	00000	INST = 0x16DB,
		INST_we = 1,
		other_ctrls = 0

2	00010	REG = 13, REG_we = 1,
		REG_sel = 1,
		other_ctrls = 0

3	00111	VAL1 = 0, VAL1_we = 1,
		RF_sel = 0,
		other_ctrls = 0

4	01010	REG2 = 27, REG2_we = 1,
		other_ctrls = 0

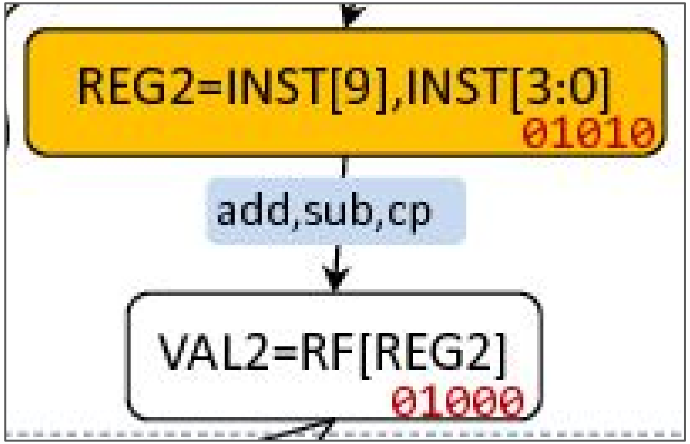
5		
6		

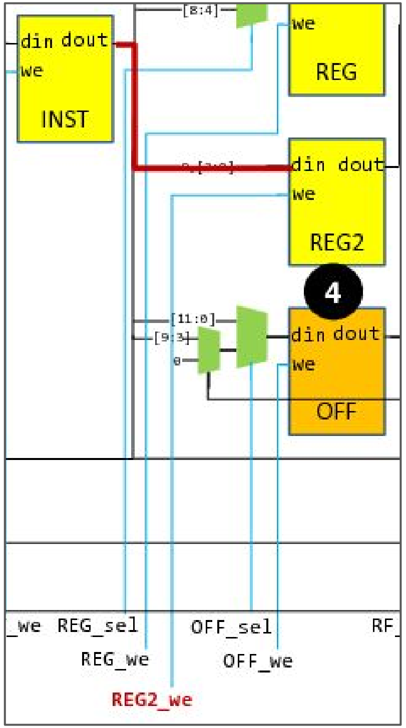
INST[15]   INST[9]   INST[3]   INST[0]

INST = INST[15:0] = 0b0001_0110_1101_1011

INST[9],INST[8:4] = 0b1,0b1011 = 0b11011

REG2 = INST[9],INST[8:4] = 0b11011 = 27





Tracing cp Cycle 5

```
; assume this is the entire program,
;   so r13, r26 and r27 is initially 0
```

Cycle	State	Changed registers/values and control signals

1	00000	INST = 0x16DB,
		INST_we = 1,
		other_ctrls = 0

2	00010	REG = 13, REG_we = 1,
		REG_sel = 1,
		other_ctrls = 0

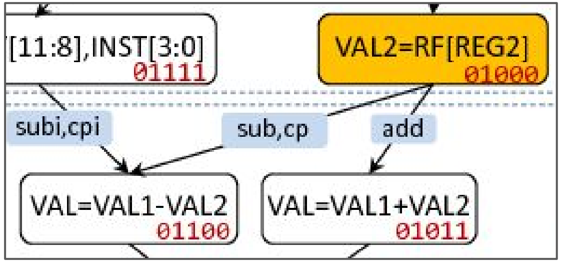
3	00111	VAL1 = 0, VAL1_we = 1,
		RF_sel = 0,
		other_ctrls = 0

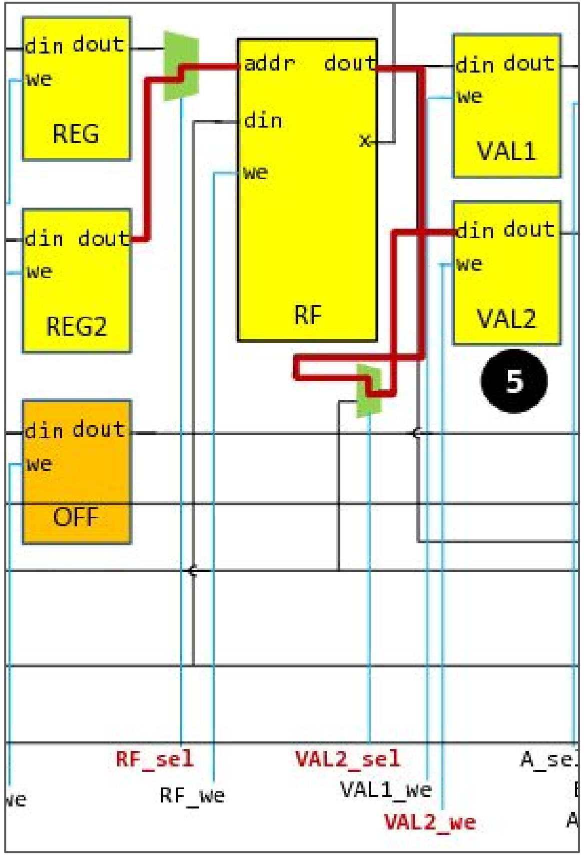
4	01010	REG2 = 27, REG2_we = 1,
		other_ctrls = 0

5	01000	VAL2 = 0, VAL2_we = 1,
		VAL2_sel = 0, RF_sel = 1,
		other_ctrls = 0

6		

```
REG2 = 27
VAL2 = RF[REG2]
     = RF[27] = 0
```





Tracing cp Cycle 6

Cycle	State	Changed registers/values and control signals

1	00000	INST = 0x16DB,
		INST_we = 1,
		other_ctrls = 0

2	00010	REG = 13, REG_we = 1,
		REG_sel = 1,
		other_ctrls = 0

3	00111	VAL1 = 0, VAL1_we = 1,
		RF_sel = 0,
		other_ctrls = 0

4	01010	REG2 = 27, REG2_we = 1,
		other_ctrls = 0

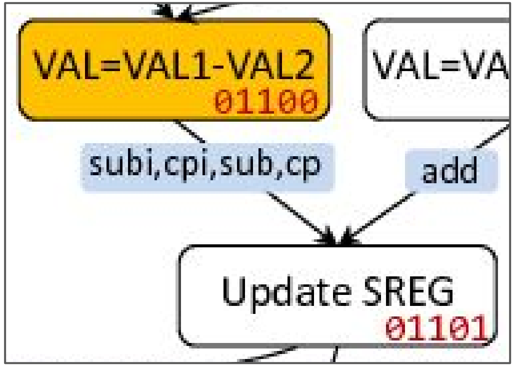
5	01000	VAL2 = 0, VAL2_we = 1,
		VAL2_sel = 0, RF_sel = 1,
		other_ctrls = 0

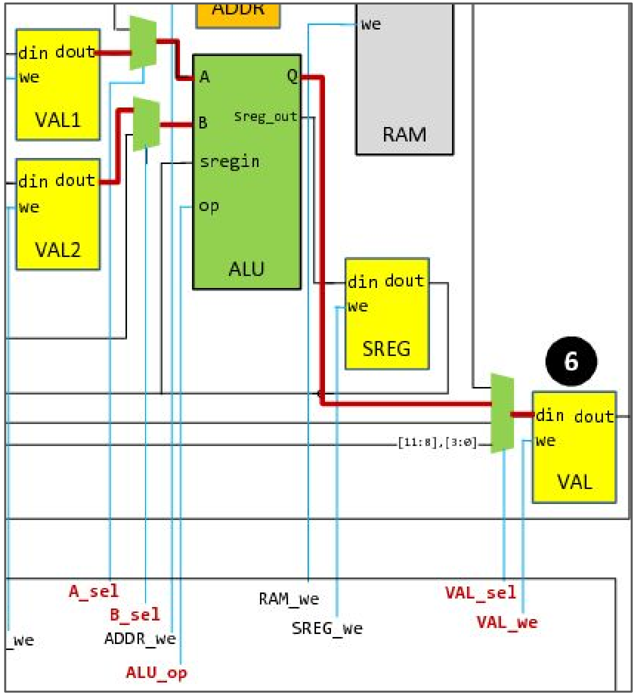
6	01100	VAL = 0, VAL_we = 1,
		VAL_sel = 1, A_sel = 1,
		B_sel = 0, ALU_op = 1
		other_ctrls = 0

VAL1 = 0, VAL2 = 0

VAL = VAL1 - VAL2

= 0 - 0 = 0





Tracing cp Cycle 7

Cycle	State	Changed registers/values and control signals
...	...	...

4	01010	REG2 = 27, REG2_we = 1,
		other_ctrls = 0

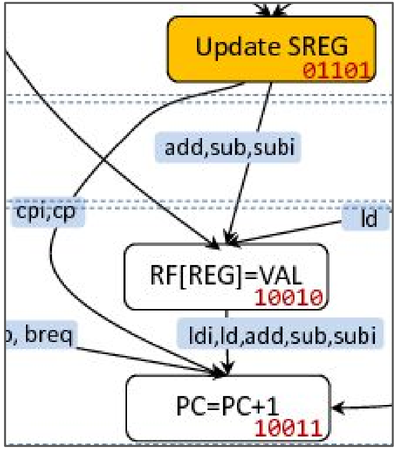
5	01000	VAL2 = 0, VAL2_we = 1,
		VAL2_sel = 0, RF_sel = 1,
		other_ctrls = 0

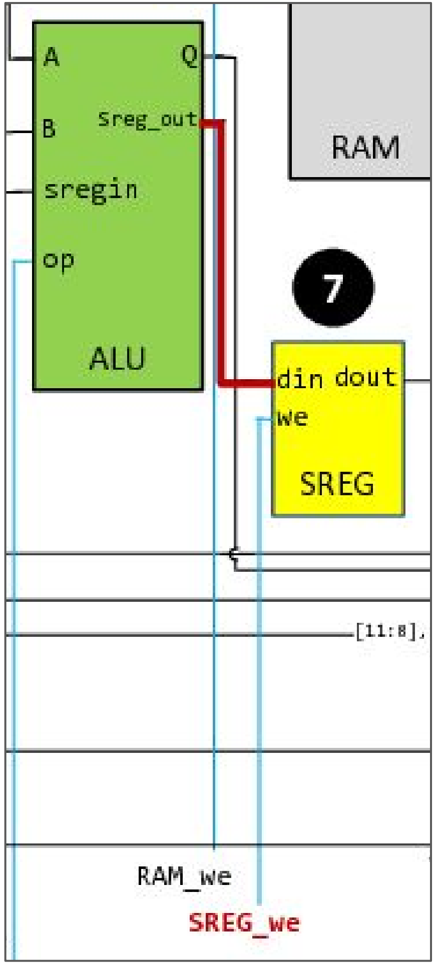
6	01100	VAL = 0, VAL_we = 1,
		VAL_sel = 1, A_sel = 1,
		B_sel = 0, ALU_op = 1
		other_ctrls = 0

7	01101	update SREG (Z=1,C=0,
		N=0),
		SREG_we = 1,
		other_ctrls = 0

8		

```
ALU: op2 - op1 = result
op2 = 0, op1 = 0, result = 0
N set if negative
N = 0 (result == 0)
Z set if result == 0
Z = 1 (result == 0)
C set if op1 < op2 (2’s comp)
C = 0 (op1 == op2 == 0)
```





Tracing cp Cycle 8

Cycle	State	Changed registers/values and control signals
...	...	...

4	01010	REG2 = 27, REG2_we = 1,
		other_ctrls = 0

5	01000	VAL2 = 0, VAL2_we = 1,
		VAL2_sel = 0, RF_sel = 1,
		other_ctrls = 0

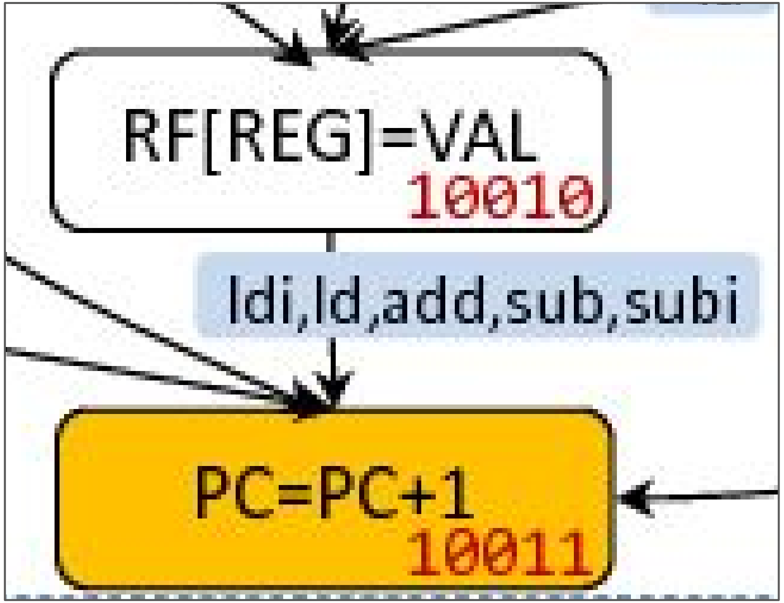
6	01100	VAL = 0, VAL_we = 1,
		VAL_sel = 1, A_sel = 1,
		B_sel = 0, ALU_op = 1
		other_ctrls = 0

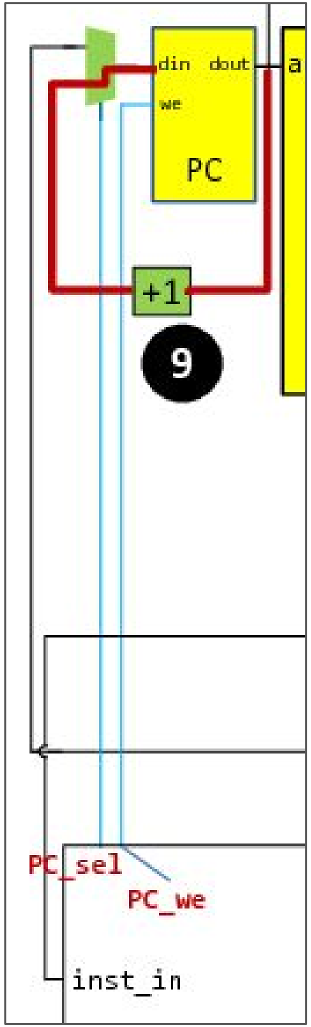
7	01101	update SREG (Z=1,C=0,
		N=0),
		SREG_we = 1,
		other_ctrls = 0

8	10011	PC = 2, PC_we = 1,
		PC_sel = 0,
		other_ctrls = 0

PC = 1

PC = PC + 1 = 1 + 1 = 2





Tracing rjmp Encoding

rjmp -23

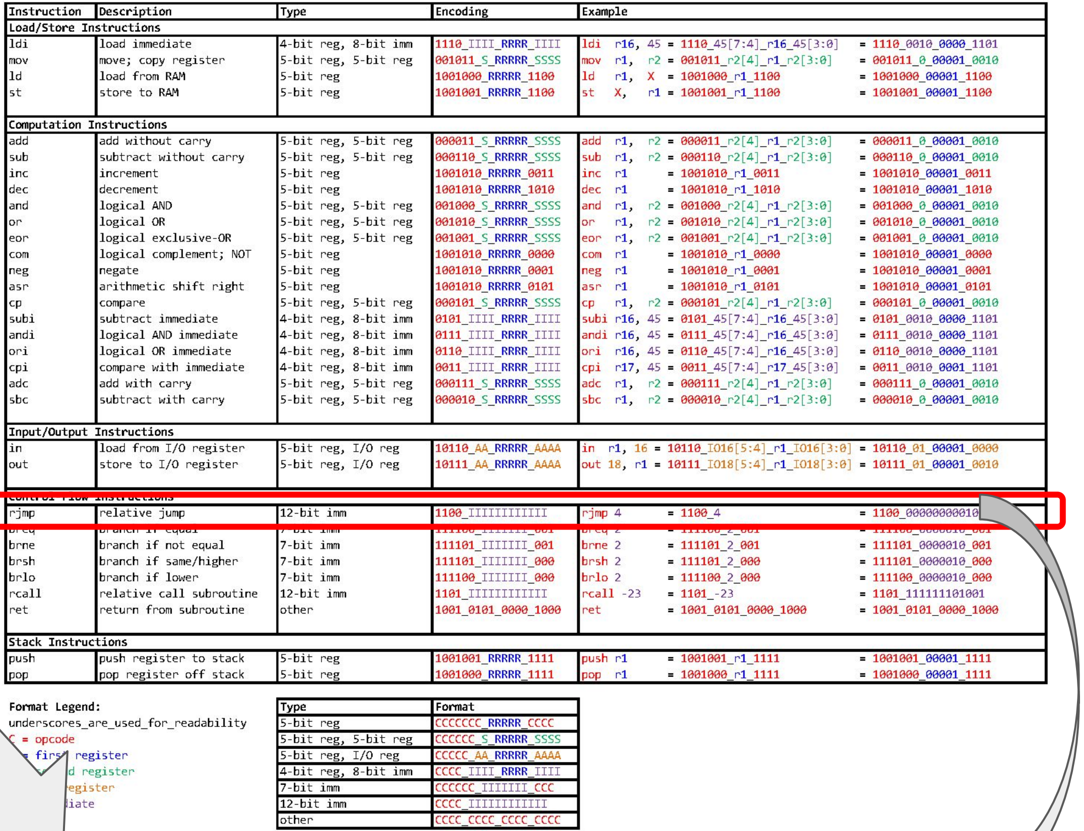
23 = 16 + 4 + 2 + 1

= 0b0000_0001_0111

invert bits and add 1 to get -23

-23 = 0b1111_1110_1000+1

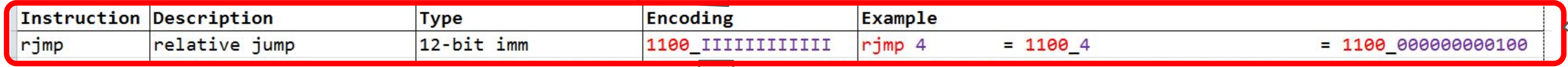
= 0b1111_1110_1001



Instruction	Description	Type	Encoding	Example
Load/Store Instructions				
ldi	load immediate	4-bit reg, 8-bit imm	1110_IIII_RRRR_IIII	ldi r16, 45 = 1110_45[7:4]_r16_45[3:0] = 1110_0010_0000_1101
mov	move; copy register	5-bit reg, 5-bit reg	001011_S_RRRRR_SSSS	mov r1, r2 = 001011_r2[4]_r1_r2[3:0] = 001011_0_00001_0010
ld	load from RAM	5-bit reg	1001000_RRRRR_1100	ld r1, X = 1001000_r1_1100 = 1001000_00001_1100
st	store to RAM	5-bit reg	1001001_RRRRR_1100	st X, r1 = 1001001_r1_1100 = 1001001_00001_1100
Computation Instructions				
add	add without carry	5-bit reg, 5-bit reg	000011_S_RRRRR_SSSS	add r1, r2 = 000011_r2[4]_r1_r2[3:0] = 000011_0_00001_0010
sub	subtract without carry	5-bit reg, 5-bit reg	000110_S_RRRRR_SSSS	sub r1, r2 = 000110_r2[4]_r1_r2[3:0] = 000110_0_00001_0010
inc	increment	5-bit reg	1001010_RRRRR_0011	inc r1 = 1001010_r1_0011 = 1001010_00001_0011
dec	decrement	5-bit reg	1001010_RRRRR_1010	dec r1 = 1001010_r1_1010 = 1001010_00001_1010
and	logical AND	5-bit reg, 5-bit reg	001000_S_RRRRR_SSSS	and r1, r2 = 001000_r2[4]_r1_r2[3:0] = 001000_0_00001_0010
or	logical OR	5-bit reg, 5-bit reg	001010_S_RRRRR_SSSS	or r1, r2 = 001010_r2[4]_r1_r2[3:0] = 001010_0_00001_0010
eor	logical exclusive-OR	5-bit reg, 5-bit reg	001001_S_RRRRR_SSSS	eor r1, r2 = 001001_r2[4]_r1_r2[3:0] = 001001_0_00001_0010
com	logical complement; NOT	5-bit reg	1001010_RRRRR_0000	com r1 = 1001010_r1_0000 = 1001010_00001_0000
neg	negate	5-bit reg	1001010_RRRRR_0001	neg r1 = 1001010_r1_0001 = 1001010_00001_0001
asr	arithmetic shift right	5-bit reg	1001010_RRRRR_0101	asr r1 = 1001010_r1_0101 = 1001010_00001_0101
cp	compare	5-bit reg, 5-bit reg	000101_S_RRRRR_SSSS	cp r1, r2 = 000101_r2[4]_r1_r2[3:0] = 000101_0_00001_0010
subi	subtract immediate	4-bit reg, 8-bit imm	0101_IIII_RRRR_IIII	subi r16, 45 = 0101_45[7:4]_r16_45[3:0] = 0101_0010_0000_1101
andi	logical AND immediate	4-bit reg, 8-bit imm	0111_IIII_RRRR_IIII	andi r16, 45 = 0111_45[7:4]_r16_45[3:0] = 0111_0010_0000_1101
ori	logical OR immediate	4-bit reg, 8-bit imm	0110_IIII_RRRR_IIII	ori r16, 45 = 0110_45[7:4]_r16_45[3:0] = 0110_0010_0000_1101
cpi	compare with immediate	4-bit reg, 8-bit imm	0011_IIII_RRRR_IIII	cpi r17, 45 = 0011_45[7:4]_r17_45[3:0] = 0011_0010_0001_1101
adc	add with carry	5-bit reg, 5-bit reg	000111_S_RRRRR_SSSS	adc r1, r2 = 000111_r2[4]_r1_r2[3:0] = 000111_0_00001_0010
sbc	subtract with carry	5-bit reg, 5-bit reg	000010_S_RRRRR_SSSS	sbc r1, r2 = 000010_r2[4]_r1_r2[3:0] = 000010_0_00001_0010
Input/Output Instructions				
in	load from I/O register	5-bit reg, I/O reg	10110_AA_RRRRR_AAAA	in r1, 16 = 10110_IO16[5:4]_r1_IO16[3:0] = 10110_01_00001_0000
out	store to I/O register	5-bit reg, I/O reg	10111_AA_RRRRR_AAAA	out 18, r1 = 10111_IO18[5:4]_r1_IO18[3:0] = 10111_01_00001_0010
Control Flow Instructions				
rjmp	relative jump	12-bit imm	1100_IIIIIIIIIIII	rjmp 4 = 1100_4 = 1100_00000000010
breq	branch if equal	7-bit imm	111100_IIIIIII_001	breq 2 = 111100_2_001 = 111100_0000010_001
brne	branch if not equal	7-bit imm	111101_IIIIIII_001	brne 2 = 111101_2_001 = 111101_0000010_001
brsh	branch if same/higher	7-bit imm	111101_IIIIIII_000	brsh 2 = 111101_2_000 = 111101_0000010_000
brlo	branch if lower	7-bit imm	111100_IIIIIII_000	brlo 2 = 111100_2_000 = 111100_0000010_000
rcall	relative call subroutine	12-bit imm	1101_IIIIIIIIIIII	rcall -23 = 1101_-23 = 1101_111111101001
ret	return from subroutine	other	1001_0101_0000_1000	ret = 1001_0101_0000_1000 = 1001_0101_0000_1000
Stack Instructions				
push	push register to stack	5-bit reg	1001001_RRRRR_1111	push r1 = 1001001_r1_1111 = 1001001_00001_1111
pop	pop register off stack	5-bit reg	1001000_RRRRR_1111	pop r1 = 1001000_r1_1111 = 1001000_00001_1111

Format Legend:
underscores_are_used_for_readability
C = opcode
= firs register
d register
egister
iate

Type	Format
5-bit reg	CCCCCCC_RRRRR_CCCC
5-bit reg, 5-bit reg	CCCCCC_S_RRRRR_SSSS
5-bit reg, I/O reg	CCCCC_AA_RRRRR_AAAA
4-bit reg, 8-bit imm	CCCC_IIII_RRRR_IIII
7-bit imm	CCCCCC_IIIIIII_CCC
12-bit imm	CCCC_IIIIIIIIIIII
other	CCCC_CCCC_CCCC_CCCC



Instruction	Description	Type	Encoding	Example
rjmp	relative jump	12-bit imm	1100_IIIIIIIIIIII	rjmp 4 = 1100_4 = 1100_000000000100

INST = 0b1100_1111_1110_1001 = 0xCFE9 = 0d53225

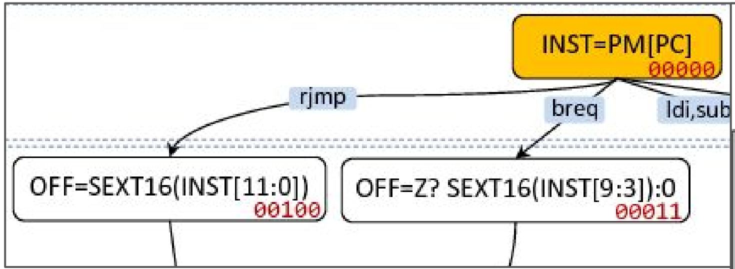
Tracing rjmp Cycle 1

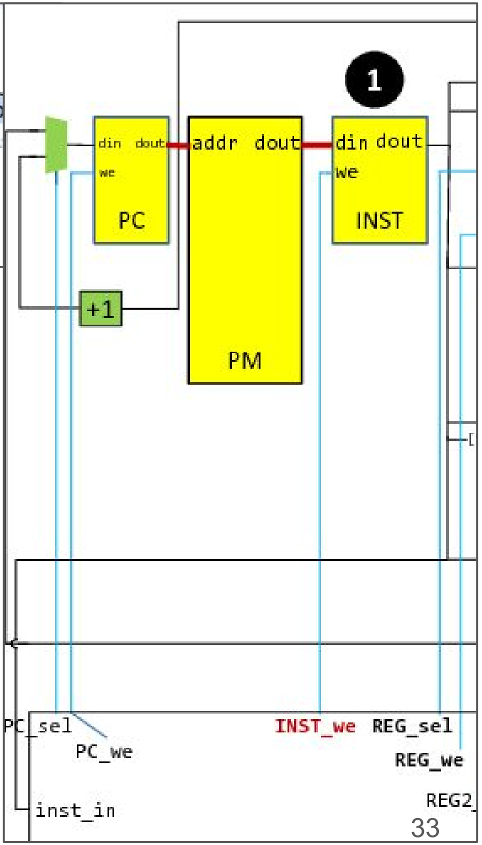
INST = 0b1100_1111_1110_1001 = 0xCFE9 = 0d53225

Cycle	State	Changed registers/values and control signals

1	00000	INST = 0xCFE9,
		INST_we = 1,
		other_ctrls = 0

2		
3		
4		
5		





Tracing rjmp Cycle 2

INST[15] INST[11] INST[0]

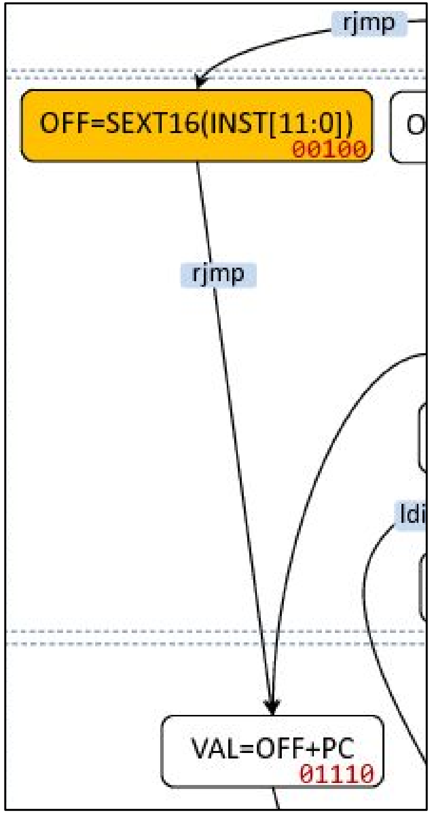
```
INST = INST[15:0]       = 0b1100_1111_1110_1001
INST[11:0]              =      0b1111_1110_1001
SEXT16(INST[11:0])) = 0b1111_1111_1110_1001 = 0xFFE9 = -23
OFF = -23
```

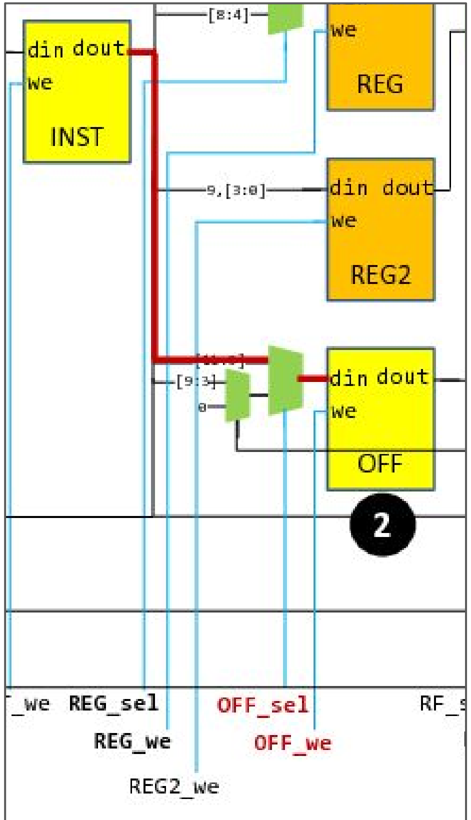
Cycle	State	Changed registers/values and control signals

1	00000	INST = 0xCFE9,
		INST_we = 1,
		other_ctrls = 0

2	00100	OFF = 0xFFE9 = -23,
		OFF_we = 1,
		OFF_sel = 0,
		other_ctrls = 0

3		
4		
5		





Tracing rjmp Cycle 3

```
if addition, then ALU_op = 0
else if subtraction, then ALU_op = 1
```

OFF = -23; PC = 2

VAL = OFF + PC = -23 + 2 = -21

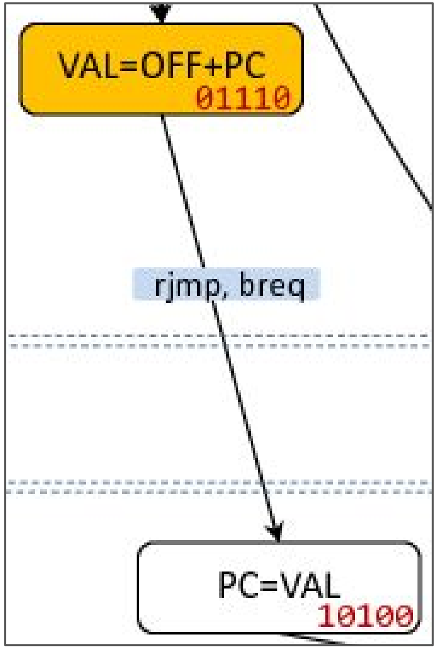
Cycle	State	Changed registers/values and control signals

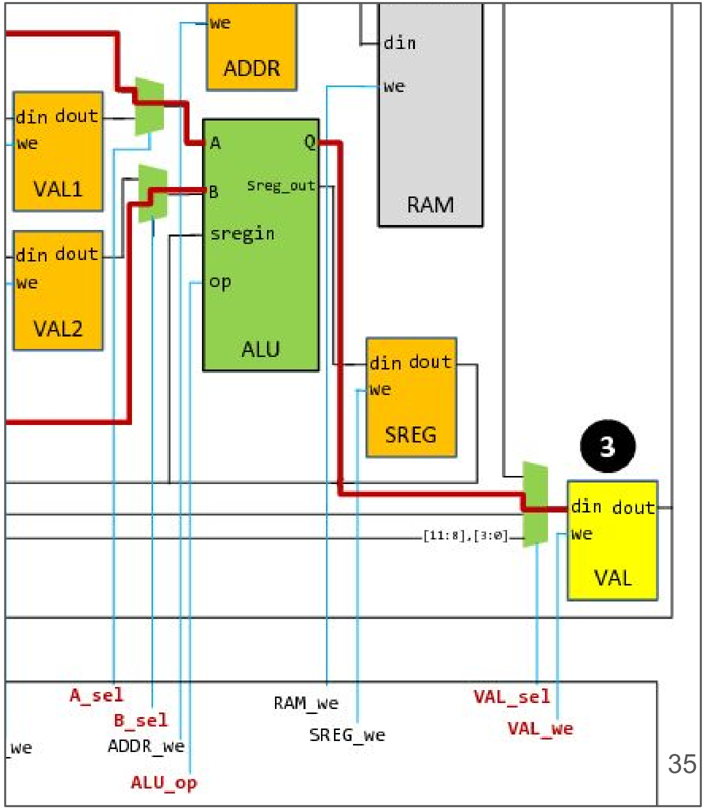
1	00000	INST = 0xCFE9,
		INST_we = 1,
		other_ctrls = 0

2	00100	OFF = 0xFFE9 = -23,
		OFF_we = 1,
		OFF_sel = 0,
		other_ctrls = 0

3	01110	VAL = -21, VAL_we = 1,
		VAL_sel = 1, A_sel = 0,
		B_sel = 1, ALU_op = 0,
		other_ctrls = 0

4		
5		





Tracing rjmp Cycle 4

Cycle	State	Changed registers/values and control signals

1	00000	INST = 0xCFE9,
		INST_we = 1,
		other_ctrls = 0

2	00100	OFF = 0xFFE9 = -23,
		OFF_we = 1,
		OFF_sel = 0,
		other_ctrls = 0

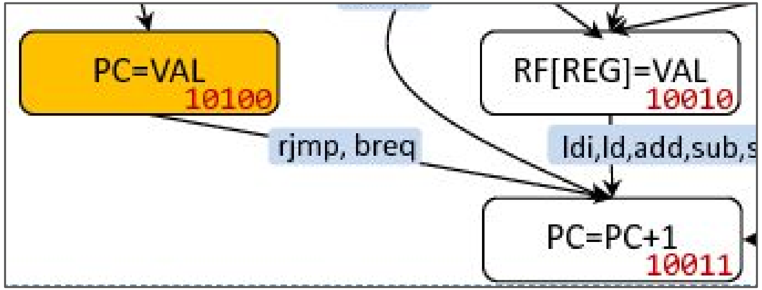
3	01110	VAL = -21, VAL_we = 1,
		VAL_sel = 1, A_sel = 0,
		B_sel = 1, ALU_op = 0,
		other_ctrls = 0

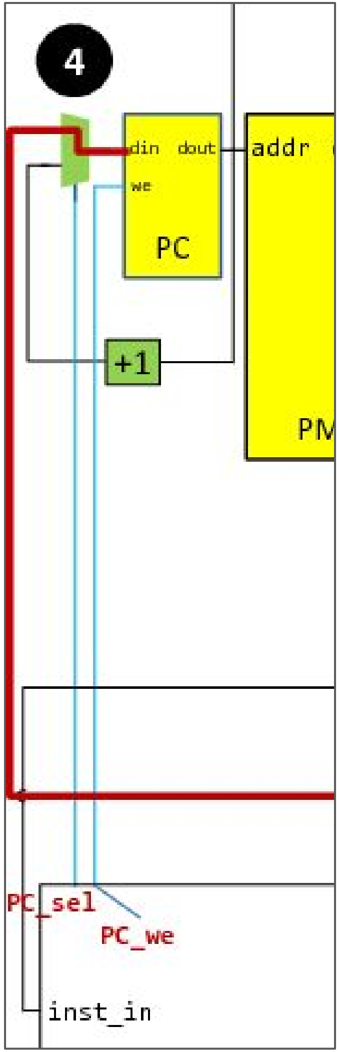
4	10100	PC = -21, PC_we = 1
		PC_sel = 0,
		other_ctrls = 0

5		

VAL = -21

PC = VAL = -21





Tracing rjmp Cycle 5

Cycle	State	Changed registers/values and control signals

1	00000	INST = 0xCFE9,
		INST_we = 1,
		other_ctrls = 0

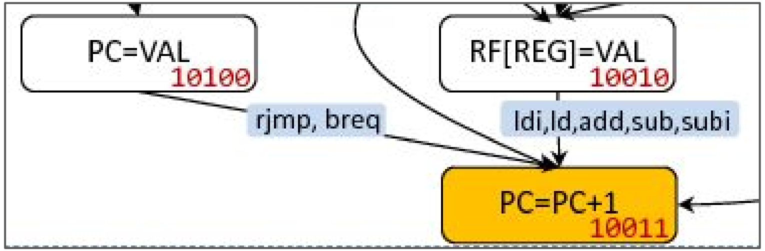
2	00100	OFF = 0xFFE9 = -23,
		OFF_we = 1,
		OFF_sel = 0,
		other_ctrls = 0

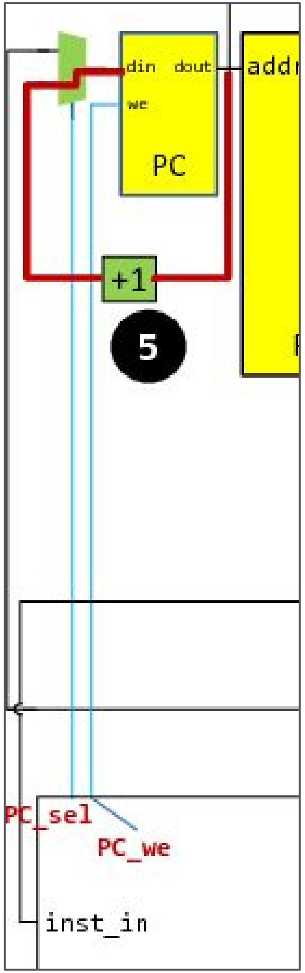
3	01110	VAL = -21, VAL_we = 1,
		VAL_sel = 1, A_sel = 0,
		B_sel = 1, ALU_op = 0,
		other_ctrls = 0

4	10100	PC = -21, PC_we = 1
		PC_sel = 0,
		other_ctrls = 0

5	10011	PC = -20, PC_we = 1
		PC_sel = 1,
		other_ctrls = 0

```
PC = -21
PC = PC + 1 = -21 + 1 = -20
```


PC=VAL
10100
RF[REG]=VAL
10010
rjmp, breq
ldi,ld,add,sub,subi
PC=PC+1
10011


din
dout
we
PC
addr
+1
5
PC_sel
PC_we
inst_in