CSSE2010/CSSE7201
Lecture 13

Instruction Set Architecture
Part 2

School of Information Technology and Electrical Engineering
The University of Queensland

Admin

- ✓ Assignment 1 marks will be released in week 8-9 when marking is completed for part 2.
- ✓ Quiz 5 is due this week Friday 10/09/21 4:00PM AEST ✓
- ✓ EX mode students – you will need Arduino based hardware items from next week for the labs.

What makes an ISA?

1. Memory models
   - A. Addressable cell size
   - B. Alignment ✓
   - C. Address spaces
   - D. Endianness ✓
2. Registers

Lecture 12

3. Instructions
4. Data types

Today

What makes an ISA? 3: Instructions

- This is the main feature of an ISA
- Instruction types include
  - **Input/Output** – communicate with I/O devices/ports
    - ("in" and "out" instructions covered previously)
  - **Load/Store** – move data from/to memory
  - ✓ **Move** – copy data between registers mov R1, R2
  - ✓ **Arithmetic & Logical** – addition, subtraction, bitwise operations
    - Dyadic (two operands, one result) add adc and ldi
    - Monadic (one operand, one result) sub sbc or eor
  - ✓ **Branching** – for deciding which instruction to perform next for if/else
- Instructions use various **addressing modes**

Sequence / selection / Repetition

AVR Instruction Examples (from previous lectures)

- **`MOV rd, rr`**
  - "move" (but actually means copy) contents of register rr (source) to register rd (destination)
  - Example: `mov r3, r14`
- **`ADD rd, rr`**
  - Add contents of register rr to register rd
- **`AND rd, rr;`   `OR rd, rr;`   `EOR rd,rr`**
  - Bitwise operations on given registers, result into rd
- **`COM rd`**
  - Flip all bits (ones' complement) in register rd
- **`LDI rd, K`** (K is a constant, -128 to 255)
  - Load constant value into a register (16 to 31)

com r5 ✓

r5 0000 1111 (*)

1111 0000

2's comp.

neg r5

1111 0001

Z N C V

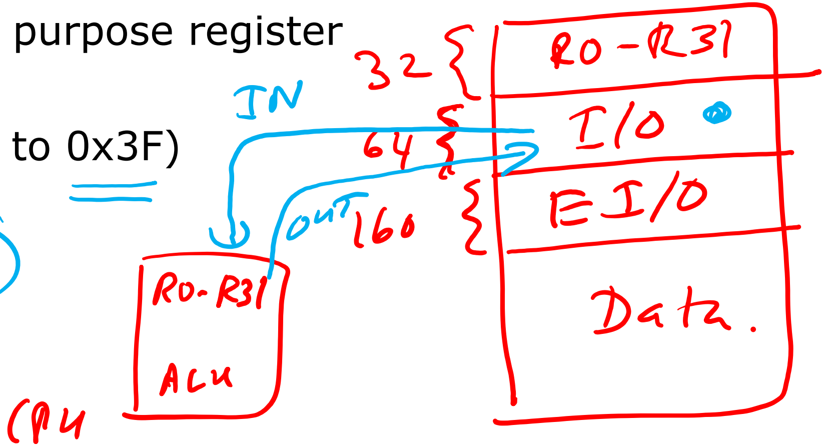
r16 - r31

IN and OUT instructions (from last lecture)

- `in rd, P`
  - Load I/O register value into general purpose register
  - rd: r0 to r31
  - P = I/O register number (0 to 63, 0 to 0x3F)
  - Example:

```
in r5, 0x26
```

IN r5, PORTA



- `out P, rr`
  - Store value from general purpose register into I/O register
  - rr: r0 to r31, P = 0 to 63 (0 to 0x3F)
  - Example:

```
out PORTD, r6
```

Data mem

SRAM

X r27 : r26
Y r29 : r28
Z r31 : r30

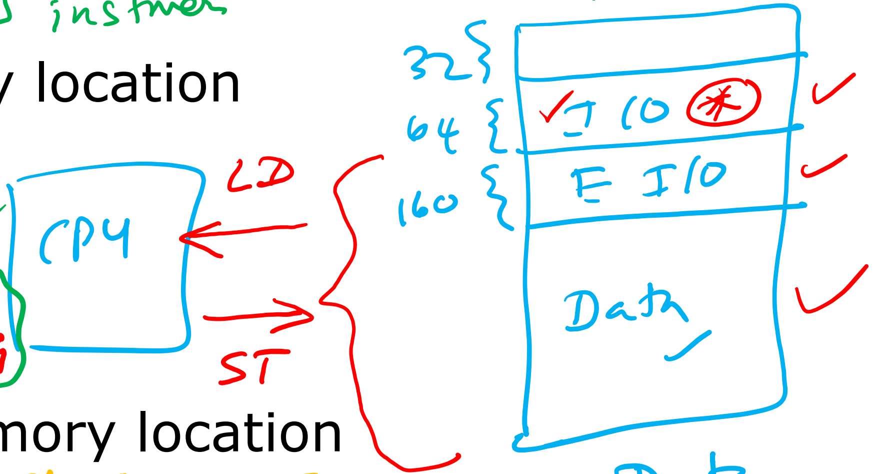
Load/store instructions

- Load
  - Load register from memory location
  - AVR Examples: `ld`, `lds`
- Store
  - Store register value in memory location
  - AVR Examples: `st`, `sts`

indirect addressing

direct addressing

lds } 32 bit instrn ⇒ 2 cells in the prog. Mem.
sts }



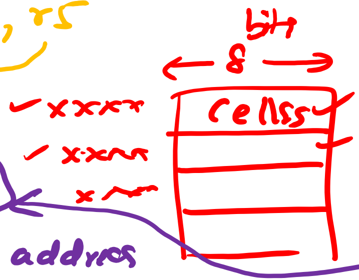
sts 0x1234, r5

ld r5, X ; r5 ← Mem[X]

st X, r5 ; Mem[X] ← r5

lds r5, 0x1234

Mem address



Data Mem

r5 ← Mem[0x1234]

Data Movement Instructions

- Data is usually **copied** not moved
- Types of data movement
  - ✓ register → register (move) mov
  - ✓ memory → register (load) ld lds
  - ✓ register → memory (store) st sts
  - ✓ I/O register → register (input) IN
  - ✓ register → I/O register (output) OUT

✓

Dyadic Instructions

- Combine 2 operands to produce a result
- Usually two types
  - Arithmetic
  - Logical
    - Bitwise

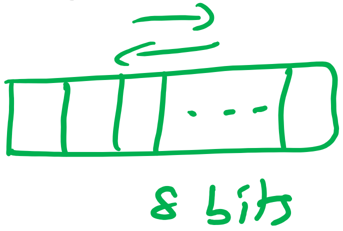
add r5, r6

opcode

operands

Monadic Instructions

- One operand → one result
- Examples
  - ✓ Setting a register (all 1’s) ser r16 → ldi r16, 0xFF
    - r16–r31
    - r16–r31
  - ✓ Clearing a register
    - clr r5 ← eor r5, r5
  - ✓ Negation (2’s complement) neg r5
  - ✓ Inversion (1’s complement negation) com r5
  - ✓ Shifting & rotation



Shift Instructions

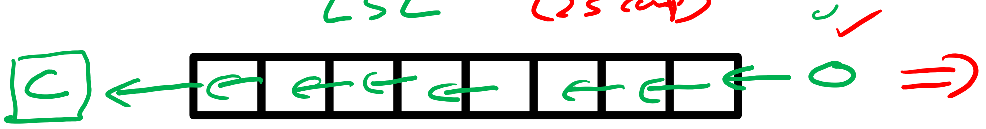
- ✓ LSL = Logical Shift Left ✓
- ✓ LSR = Logical Shift Right → unsigned ÷2
- ✓ ASR = Arithmetic Shift Right → signed ÷2 (2's comp)

lsl vs

0000 0011 ⇒ 3
0000 0110 ⇒ 6

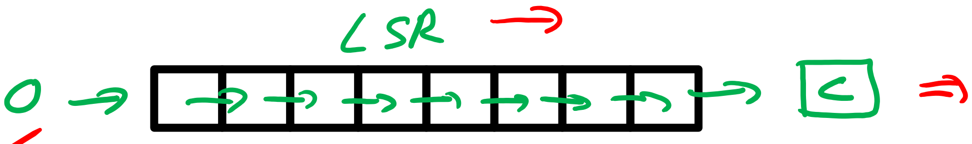
Monadic

carry flag in status reg. →

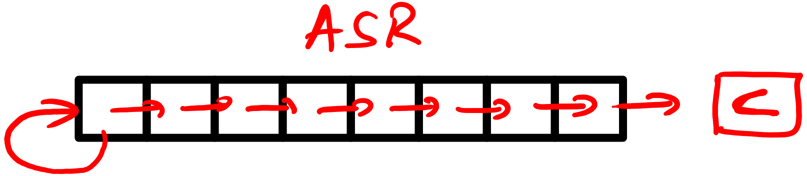


⇒ Mutiply × 2
× 2^n (LSL n bits)

unsigned →



⇒ division by 2
÷ 2^n (LSR n bits)



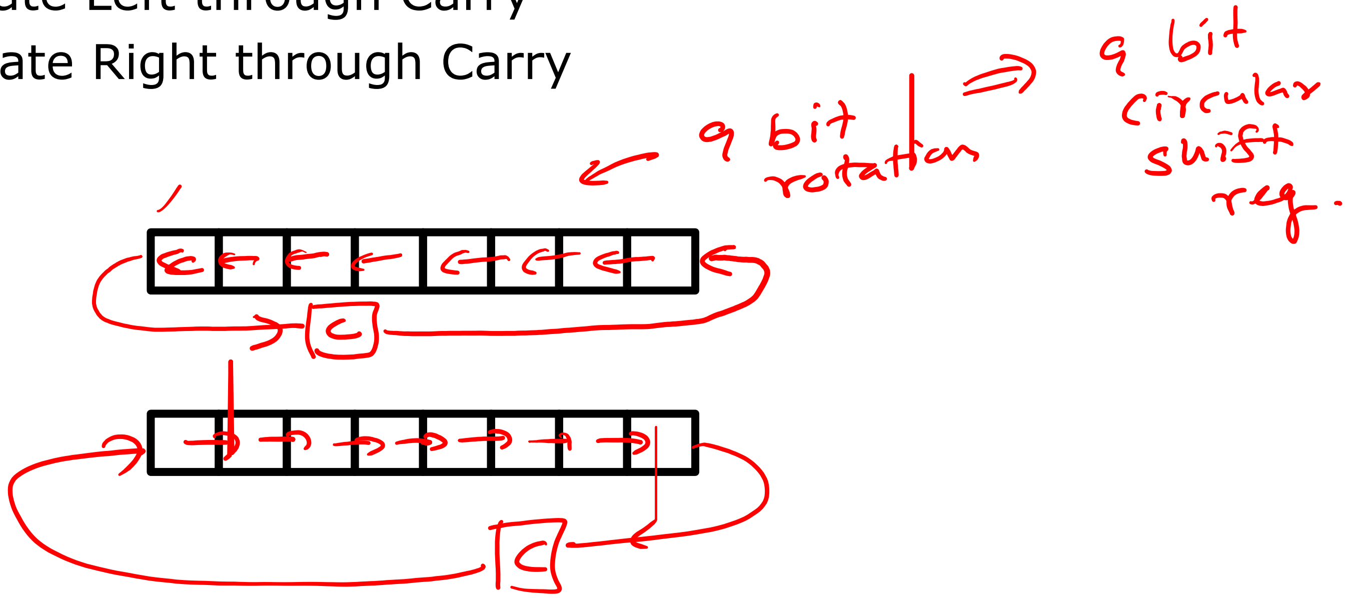
4 bits 2's com

1000
↓asr
1100

-8 ÷2
↓
-4

Rotate Instructions

- ROL = Rotate Left through Carry
- ROR = Rotate Right through Carry

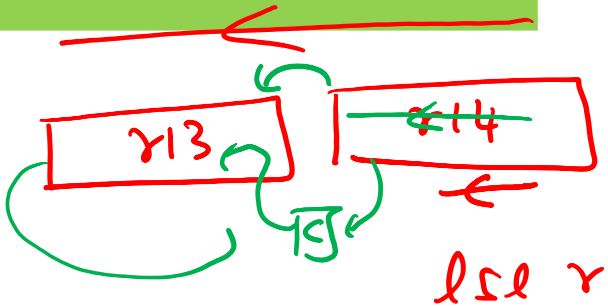


What AVR instruction should follow `lsl r14` to perform a multiplication by 2 of the 16-bit *unsigned* value stored in r13:r14?

- 25% A. `asl r13`
- 25% B. `lsl r13`
- 25% [✓ C. `rol r13`]
- 25% D. `rol r14`

16 bit unsigned.

16 bit



lsl r14

rol r13

x2

10

Comparison Instructions

- Same as subtraction
  - Result not saved
  - Sets status register flags (Z, N, C, V …)
- Instruction reference specifies which status register bits are set by which instructions

✓ CP r5, r6 ← (r5 − r6) ✓

↳ set the flags.

z = 1

r5=10
r6=10

10 − 10 = 0

Branch instructions

- Based on status register values
- Example:
  - **breq label**
    - Branch if Z=1
  - **brne label**
    - Branch if Z=0

prog. mem address.

```
cp r5, r6;
breq next;
```

Z=1

```
next :
```

PC

jmp label

jmp
rjmp

Conditional branches

Addressing Modes

Instruction operands can come from different places

- **Immediate** addressing:
  - An operand value is in the instruction
    - AVR Example: `andi r17, 0xBA`

ldi r5, 0x22

- **Register** addressing:
  - Only register numbers in instruction
    - AVR Example: `and r18, r19`
- **Direct** addressing:
  - Memory address is in instruction

= 0x

    - AVR Example: `lds r15, $1234`
  - Note, these AVR instructions occupy two instruction words (32 bits)

Addressing Modes (cont.)

- **Indirect** addressing
  - Memory address in register
  - Register number is in instruction
    - AVR Example: `ld r5, X`

- These addressing mode names are as per Tanenbaum “Structured Computer Organization”
- AVR instruction set manual uses different names

What makes an ISA? 4: Data Types

- Numeric
  - Integers of different lengths (8, 16, 32, 64 bits)
    - Possibly signed or unsigned
  - Floating point numbers, e.g. 32 bits (single precision) or 64 bits (double precision)
  - Many others not listed

- Non-numeric
  - Boolean (0 means false, 1 means true) – stored in a whole byte or word
  - Bit-map (collection of Booleans, e.g. 8 in a byte)
  - Characters
  - Pointers (memory addresses)

4: Data types (cont.)

- Different machines provide *hardware support* for different data types.
  - e.g. Intel-64 Architecture:

Data Type	8 bits	16 bits	32 bits	64 bits	128 bits
Signed integer	✓	✓	✓	✓	
Unsigned integer	✓	✓	✓	✓	
BCD integer	✓				
Floating point			✓	✓	

  - e.g. Atmel AVR:

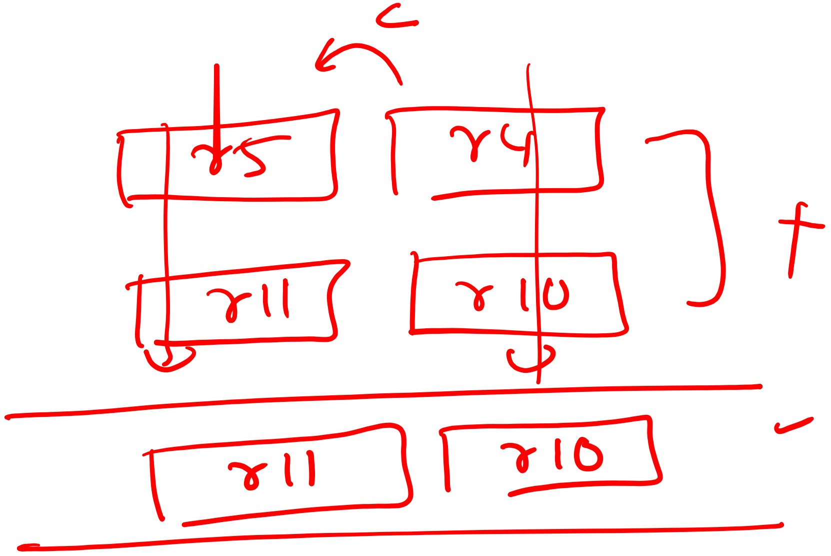
Data Type	8 bits	16 bits	32 bits	64 bits	128 bits
Signed integer	✓				
Unsigned integer	✓				
BCD integer					
Floating point					

4: Data types (cont.)

- Other data types can be supported through software!
  - e.g. 16-bit integer operations can be built out of 8-bit operations, using a sequence of instructions.

AVR Example of 16-bit addition

- Add 16-bit quantity r5:r4 to r11:r10. (Result should be in r11:r10.)



add r10, r4

adc r11, r5

r11 ← r11+r5+$\boxed{c}$

Summary:
What makes an ISA?

- Memory models
- Registers
- Instructions
- Data types

- If you know all these details, you can
  - Write machine code that runs on the CPU
  - Build the CPU