1. AVR ISA & Programming

COMP2121 • KC Notes

1.1 Atmel AVR

- 8 bit Reduced Instruction Set Computer (RISC) microprocessor architecture
  - 16 bit length instructions with 1 clock cycle execution
  - Load-store memory access architecture
  - Calculations are on registers
  - Two stage instruction pipelining
  - Internal program memory and data memory
  - On-chip peripherals: digital I/O, ADC, EEPROM, UART, PWM
- **Registers:**
  - 32 8-bit registers r0 to r31
    - Grouped into r0 to r15 and r16 to r31
    - Some instructions only work on second group due to instruction encoding
  - General purpose registers:
    - X, r27:r26
    - Y, r29:r28
    - Z, r31:r30, above three work as address indexes/pointers
    - r0 stores data loaded from program meory
    - r1:r0 stores multiplication instruction
  - I/O registers
    - 64+416 8-bit registers defined in m2560def.inc file
    - **Stores data or addresses**, used to **control signal bits**

1.2 Status Register

- **SREG**, the status register: contains information about the result of the **most recently executed arithmetic instruction**, updated after any **ALU (arithmetic logic unit) operation**
  - Allows program flow **to be altered based on conditional operations**
  - Not automatically stored in interrupts, must be handled in software

Bit	Flag, bit name	Usage
7	I: Global Interrupt Enable	Enables and disables interrupts. Cleared after interrupt occurs and set when RETI is called.
6	T: Bit Copy Storage	Bit Copy instructions bld and bst use T-bit as source and destination.
5	H: Half Carry Flag	Half carry flag, useful for BCD arithmetic when you carry from bit 4

4	S: Sign Bit	XOR between N and V
3	V: Two's Complement Overflow Flag	Useful in two's complement arithmetic
2	N: Negative Flag	Most significant bit of the result
1	Z: Zero Flag	Zero result in an ALU operation
0	C: Carry Flag	For x+y, it is the carry from most significant bit For unsinged x-y, indicates if x < y

1.3 AVR Address Spaces

- **Data memory**
  - Stores data to be processed
  - Covers **register file** (0x0000-0x001F), **I/O registers** (0x001F-0x0200) with I/O and memory addresses and external **SRAM data memory** (0x0200-RAMEND)
  - 8 bits wide from 0x0000 to RAMEND
- **Program memory**
  - Stores program and constants
  - 16 bit flash memory that is **read only** and **non-volatile** (instructions are retained when power is off)
  - Accessed using lpm and spm
- **EEPROM memory**
  - Large permanent data storage accessed using load and store instructions with special control bit settings
  - 8 bit

1.4 AVR Instruction Format and Classes

- Formatted in 1 word (16 bit) long instructions, e.g. add, sub, mul, brge
  - Some are 2 words (32 bit) long, e.g. lds
  - See instruction set
- **Arithmetic and Logic**
  - Arithmetic: add, sub, inc, dec, mul
  - Logic: and, or
  - Shift: lsl
- **Data transfer**
  - GP register: mov
  - I/O registers: in, out
  - Stack: push, pop
  - Immediate values: ldi
  - Data memory: ld, st, lds, sts
  - Program memory: lpm
- **Program control**
  - Conditional branches: breq, sbic
  - Unconditional branches: rjmp
  - Call/return subroutine: rcall, ret
- **Bit and others**
  - Set bit: sbi
  - Clear bit: cbi
  - Copy bit: bst
  - Others: nop, break, sleep, wdr

1.5 AVR Addressing

- Immediate (0x0F)
- Register direct (r0, PINA)
- Memory related addressing mode
    - Data memory: direct (0x0F), indirect (X), indirect with displacement (Y+10), pre decrement (-Y), post increment (Y+)
    - Program memory: direct (0x0000), relative (rjmp k), indirect (icall), constant addressing (lpm), post increment (Z+)

1.6 Specific samples

Compare 16-bit numbers

```
cpi r24, low(244)
ldi temp, high(244)
cpc r25, temp
brne halt
```

1.7 Assembly directives

- Assembly program consists of **assembler directives** and **executable instructions**
- Assembler directives:
    - .equ symbol = 2 — set a symbol equal to a constant value
    - .set symbol = 2 — set a symbol to a value
    - .def input = r17 — set an alias for a register
    - .dseg — start data segent
    - .cseg — start code segment (default)
    - .eseg — start EEPROM segment
    - ConstantString:
      .db 3,1,4,5 — program memory – stores **byte** constant
    - ConstantWord:
      .db 3, 2958, 32000 — program memory – stores **word**, little endian
    - InventoryCount:
      .byte 9 — data memory – stores number of bytes
    - .include “m2560def.inc” — include a file
    - .exit — Stop processing assembly file
    - .macro
      .endmacro — Begin and end a macro
    - .org 0x0000 — Tells the assembler the starting address of a segment

1.8 Memory access and memory mapping

- Constants are stored in **flash memory** (program memory)
    - Program memory data is packed into words. Data is stored with the low byte on the left (little endian)
    - Accessing it using a pointer must be done by shifting the pointer to the left once to access each 8 bits, rather than each word
- Variables are allocated and stored in SRAM (data memory)

1.9 Assembly expressions

- Operators:
    - << shift left
    - >> shift right
- Functions:
    - low(label) low byte of an expression
    - high(label) high byte of an expression
- Macros:
    - Sequence of instructions that is repeated several times
    - Is like an **inline function** in C – when assembled, the definition is expanded where it is called
    - Can take parameters @0 to @9

1.10 Assembly

- **Absolute assembly**: code (e.g. .asm file) is assembled and **a loader** transfers machine code to the system's computer memory
- **Relocatable assembly**: source files are assembled separately into object files, and **a linker** resolves unresolved addresses and makes the object files into a single executable file
- Assembly process: first pass changes all macros, and symbols into their values, second pass assembles the code.

1.11 Stack

- **Stack**: LIFO, is **a block of consecutive bytes in SRAM memory** with stack pointer
    - Grows from **highest address to lower address**
    - Stack pointer **SP** is default at 0x000 and needs to be initialised
    - push and pop operations

1.12 Functions

- **Stacks and functions**: **conflict registers** are pushed (saved) into the stack and then popped after the function
    - General registers can also be used to store actual parameters
    - Functions create **stack frame** on the stack with a **stack frame pointer**
    - **Stack frames** contain **return address, conflict registers, parameters (arguments) and local variables**
- **Caller**: before calling a function, parameters are stored in designated registers
- **Call the callee** e.g. using instructions rcall, icall or call
    - **Prologue**: store conflict registers, stack frame register by using push
        - Update stack pointer and stack frame pointer
        - Pass actual parameters to formal parameters on the stack
    - **Function body**: tasks the function does
    - **Epilogue**: store return values, deallocate the stack frame
        - out the stack pointer
        - restore conflict registers using pop
        - Return to the caller by using ret

```
push YL           ; store conflict registers
push YH
push r19
in YH, SPH        ; initialise stack frame pointer value
in YL, SPL
push YH           ; save pointer in stack
push YL
```

1.13 Recursive Functions

- Recursive functions call themselves
- Use **call trees** to find the maximum size a recursive function uses up (find the **longest weighted path** in the call tree)