AVR

Today
- ATmega328p chip
- AVR assembly especially for PA3ifdots.java
AVR Instruction Set Architecture, or Assembly

ATmega328p

Why assembly?

AVR ISA

Handling GetButton and SetPixel calls, (Calling Convention)

Handling if statements (Condition Codes and Branches)

Handling expression evaluation (Operations and Stack instructions)

Variables on the stack and in the heap
**ATmega328p**

**Terminology**
- Atmel, a company
- AVR, 8-bit RISC instruction set architecture for a microcontroller
- ATmega328p, AT for Atmel, MegaAVR microcontroller, 32kb flash, 8-bit AVR, p=low power
- Arduino, programming environment for various boards with some AVR chips

**Uses**
- Very popular for hobbyists
- Industry: Whirlpool appliances, electric car charger, medical products, …
Why Assembly?

It is the target language for (C++, MeggyJava) compilers,
– so can generate symbolic code, and
– do not need to resolve (references to) labels,
– also don’t have to link to create .hex files

We can link with the C++ run time Meggy Jr libraries

Assembly programming:
– For some embedded processors, still need to do some assembly programming (e.g. device drivers).
– We want to understand / express how the run-time stack works.
AVR Instruction Set Architecture (ISA)

AVR is an 8-bit (byte) Harvard RISC Architecture
- Two 8-bit words (and register pairs e.g. R0, R1) can be interpreted as 16 bits ints

Harvard: There are separate spaces
- data space (data) (0-RAMEND)
- program space (text) (0-FLASHEND)

There are 32 Registers, organized in a register file R0 – R31
There is a run time Stack (stack pointer/ push / pop)

RISC: Reduced Instruction Set, What does it mean?
- Only load/store instructions can access the memory
- Most instructions work on registers only and have therefore fully predictable timing (#clocks to execute)
- No self-modifying code.
Execution Model

PC →

- text
  - ldi ...
  - add ...
  - sub ...

- data
  - heap
  - stack

- Registers
  - r0
  - r1
  - r2
  - r3
  - r31

- Stack Pointer
  - r29:r28

ALU
Addressing modes

Program and data addressing modes support access to the Program (flash) and Data memory (SRAM, Register file, I/O memory). See the AVR instruction Set document for the details.

Instructions are packed in one or two words (2 bytes).

- **Direct register** uses the (names of) registers as operands
- **Data direct** has a 16-bit data address in the word following an instruction word
- **Relative** (PC relative) adds an offset to the program counter. The offset has a limited range (-2048..2047)
Meggy Java program for translation to AVR (calls)

/**
 * PA3ifdots.java
 *
 * An example for the students to code up in AVR assembly for PA1.
 * The language features will be from the PA3 grammar.
 */

import meggy.Meggy;

class PA3ifdots {

    public static void main(String[] whatever) {
        if (Meggy.checkButton(Meggy.Button.Up)) {
            Meggy.setPixel( (byte)3, (byte)(4+3), Meggy.Color.BLUE );
        }
        if (Meggy.checkButton(Meggy.Button.Down)) {
            Meggy.setPixel( (byte)3, (byte)0, Meggy.Color.RED );
        }
    }
}

Calling convention

Calling convention is interface between caller and callee
- callers have to pass parameters to callee
- callees have to pass return values to caller
- callers and callees save registers
  caller saves registers r18-r27, r30-r31
  callee saves registers r2-r17, r28-r29
- Arguments - allocated left to right, r25 to r8
  r24, r25    parameter 1, only use r24 if just a byte parameter
  r22, r23    parameter 2
  ...  r8, r9    parameter 9

Return values
  8-bit in r24, 16-bit in r25:r24,
  up to 32 bits in r22-r25, up to 64 bits in r18-r25.
Meggy Java program for translation to AVR (calls)

/* PA2bluedot.java */
import meggy.Meggy;

class PA2bluedot {
    public static void main(String[] whatever){
        Meggy.setPixel( (byte)1, (byte)2, Meggy.Color.BLUE );
    }
}

/* prologue: function */
.data
    .file  "PA2bluedot.cpp"  /* frame size = 0 */
    __SREG__ = 0x3f
    __SP_H__ = 0x3e
    __SP_L__ = 0x3d
    __CCP__ = 0x34
    __tmp_reg__ = 0
    __zero_reg__ = 1
    .global __do_copy_data
    .global __do_clear_bss
    .text
    .global main
    .type  main, @function

main:  /* prologue: function */
    call _Z18MeggyJrSimpleSetupv
    ldi r24,lo8(1)
    ldi r22,lo8(2)
    ldi r20,lo8(5)
    call _Z6DrawPxxhh
    call _Z12DisplaySlatev
    jmp  .L2
    .L2:  /* prologue: function */
    .size  main, .-main
/**
 * PA3ifdots.java
 * 
 * An example for the students to code up in AVR assembly for PA1.
 * The language features will be from the PA3 grammar.
 */

import meggy.Meggy;

class PA3ifdots {

    public static void main(String[] whatever){
        if (Meggy.checkButton(Meggy.Button.Up)) {
            Meggy.setPixel( (byte)3, (byte)(4+3), Meggy.Color.BLUE );
        }
        if (Meggy.checkButton(Meggy.Button.Down)) {
            Meggy.setPixel( (byte)3, (byte)0, Meggy.Color.RED );
        }
    }
}
### AVR Status Register

Status Register (SREG) keeps some bits (flags) that represent an effect of a previously executed instruction.

Some important flags (there are more, check the Atmel AVR manual):

- **C**: Carry flag, a carry occurred (bit overflow)
- **Z**: Zero flag, result was 0
- **N**: Negative flag, result was negative

The effect on flags by instruction execution can be cleared (0), set (1), unaffected (-)

Conditional Branch instructions (breq, brlo, brlt, brne) use these flags:

- `brne label`
The comparison and arithmetic instructions set the flags \((Z,N,C,\ldots)\)

Comparison instructions: \texttt{cp} \texttt{ cpc} \texttt{tst}

Arithmetic instructions:
\texttt{adc} \texttt{add} \texttt{sbc} \texttt{sub} \texttt{neg} \texttt{and} \texttt{or} \texttt{eor} \texttt{lsl} \texttt{lsl} \texttt{muls} \texttt{rol} \texttt{ror}

Conditional branch instructions inspect the flags:
Branch instructions: \texttt{brlo} \texttt{brlt} \texttt{brmi} \texttt{brne}

Branches branch PC relative and have a limited range \((-64 .. 63)\)
Therefore, if we don’t know how far a branch will branch, we need to branch to a jump instruction (jmp), which can reach all instructions
/* PA5movedot.java */

if (Meggy.checkButton(Meggy.Button.Up)) {
    this.movedot(curr_x, (byte)(curr_y+(byte)1));
    Meggy.toneStart(localvar, 50);
} else {}