AVR and MeggyJr Simple

Today
- Finish AVR assembly especially for PA3ifdots.java
- Meggy Jr Simple run-time library
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
/**
 * 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 );
        }
    }
}
Arithmetic: bytes and ints

AVR is an 8 bit architecture, but has support for 16 bit ints.

This is accomplished by having register pairs, and having certain instructions taking certain flags into account:

```
# add r1:r0 to r3:r2
add r2,r0  # Rd = Rd + Rr  sets C
adc  r3,r1  # Rd = Rd + Rr + C
```

Subtraction: check out “sub” and “sbc”
Multiplication: check out “muls”
Bitwise AND: check out “and”
Meggy Java program for translation to AVR (expression eval)

/* PA5movedot.java */
...
return ((byte)(0-1) < x) ...

.file "PA5movedot.java"
# Load constant int 0
ldi r24,lo8(0)
ldi r25,hi8(0)
# push two byte expression onto stack
push r25
push r24

# Load constant int 1
ldi r24,lo8(1)
ldi r25,hi8(1)
# push two byte expression onto stack
push r25
push r24

# load a two byte expression off stack
pop r24
pop r25

# Do INT sub operation
sub r24, r18
sbc r25, r19
# push hi order byte first
# push two byte expression onto stack
push r25
push r24

# Casting int to byte by popping
# 2 bytes off stack and only push low bits
# back on. Low bits are on top of stack.
pop r24
pop r25
push r24
Variables on the Stack and Heap

PC → text

ldi ...
add ...
sub ...

PC → data

heap

stack pointer ← r29:r28

stack

Registers

r0
r1
r2
r3
r31

ALU
Stack and heap

**Stack pointer:**
points at first available location on the run time stack
varies during expression evaluation

**Frame pointer:**
a fixed pointer in the stack frame so that parameters and local
variables can be associated with an offset from the frame pointer

**Allocating space on the heap with `malloc` library function:**
`malloc` allocates \( n \) consecutive bytes in the heap and
returns the address of the first byte allocated.
(Will see examples of this later).
Data Indirect addressing

Some register pairs are used for indirect addressing.
There are special names for these Indirect Address Registers

\[ X=R27:R26, \quad Y=R29:R28, \quad Z=R31:R30 \]

\begin{verbatim}
in r28, __SP_L__    // putting the stack pointer into r29:r28
in r29, __SP_H__

ldd r24, Y+3      // load byte that is 3 bytes from address in r29:r28
                  // r24 = M[r29:r28 + 3]

std Y+1, r24      // store value in r24 to address r29:r28+1
                  // M[r29:r28 + 1] = r24
\end{verbatim}

There are pre-decrement and post-increment indirect addressing modes for data structure (Stack) manipulation

The run time stack is implicitly manipulated with (push) and (pop) instructions, SP is the name of the stack pointer
Meggy Jr Simple Library

Key concepts

- LED screen (pixels)
- Auxiliary LEDs
- Buttons
- Speaker

- Check the AVR-G++ generated code for library calls, and their calling sequence. AVR-G++ (and also MeggyJava) links in run time libraries:
- **Meggy Jr** Library provided an interface to set and read values in the Display Memory
- **Meggy Jr Simple** lies on top of Meggy Jr library, and provides a higher level API with names for e.g. colors
- Michelle Strout and students (honors projects / theses) added some functionality to the Meggy Jr Simple library
Meggy Jr Simple Library functions

ClearSlate() -- erase the whole slate
DrawPx(x,y,color) -- set pixel (x,y) to color
DisplaySlate() -- copy slate to LED Display Memory
SetAuxLEDS(value)
   -- 8 LEDs above screen numbered 1, 2, 4,..,128 (left to right)
   value is a byte encoding in binary which LEDs are set
   SETAuxLEDS(53) sets LEDs 1, 4, 16, and 32
ReadPx(x,y) -- returns byte value of pixel (x,y)

CheckButtonsDown()
   -- sets 6 variables: Button_(A|B|Up|Down|Left|Right)
GetButtons() returns a byte (B,A,Up,Down,Left,Right: 1,2,4,8,16,32)
ToneStart(divisor, duration)
   -- starts a tone of frequency 8 Mhz/divisor for ~duration milliseconds
   There are predefined tones.

Check out MeggyJrSimple.h
Example AVR-G++ program

/*  1/24/11, MS, goal is to exercise all of the routines in MeggyJrSimple */
#include "MeggyJrSimple.h"
#include <util/delay.h>

int main (void) {
    MeggyJrSimpleSetup();
    DrawPx(0, 1, Red);    // should display red LED
    DisplaySlate();
    // If <0,1> pixel is red, set auxiliary light
    if (ReadPx(0,1)==Red) {  SetAuxLEDs (4); }
    while (1){
        CheckButtonsDown();
        if (Button_A) {   Tone_Start(ToneC3, 1000); }
        if (Button_B) {   SetAuxLEDs(16); }
        if (4 & GetButtons()) { SetAuxLEDs(31); }  //
        if (Button_Up) { delay_ms(256); }
    }
    return 0;
}
Mapping Meggy Java Interface to Meggy Simple Interface

Let’s look at some examples of how this works.
Before Next Time

Finish PA1 and try checking it in online. It is due in 5 days!

Check out the discussion board on Canvas.