Making Connections...

- Functions
  - Do they “own” all the CPU’s registers?
  - Argument passing
    - How? (Value/Reference)
    - Where? (Global/Register/Stack)
- Mixing C++/ASM
  - Why?
  - How?

1

Assembly language style

- Similar to C++
- Program structure
  - Effectively use functions
  - Code reuse
- Documentation
  - Design
  - Function arguments and return values
  - Preconditions / postconditions
- Code is longer than C++
  - Easy to get lost
  - Harder to follow
  - Break down into smaller functions
- Coding
  - It is sometimes easier to code in C++, then convert
  - Or use a C++ compiler

2

Functions and arguments

- We have seen the wait function, which saves all registers.
- Is there another way to do this?

3

Register saving (1/2)

- Function saves registers used
  - Function knows the registers used (sometimes said to be “destroyed”)
  - Function does not know the registers the caller is using, so must save all destroyed
- Caller saves registers
  - Caller knows registers in use
  - Caller does not know registers destroyed by function, so must save all in use

4

Register saving (2/2)

- What does gcc 3.3.5 for the HC11 do?
  - Assumes that D, X, Y, and CCR are clobbered.

5

Argument passing

- “Pass” in global
  - Usually values
- Pass in registers
  - Values
  - References
- Pass on stack
  - Values
  - References

6

"Pass" in global variable

```
point: .space 3
sum:   .space 1

_start:
   ldaa #3
   ldx #point;point to
   data
   staa 0,x  ;save it
   ldaa #2   ;load 2
   staa 1,x  ;save it
   ldaa #1   ;load 1
   staa 2,x  ;save it
   jsr calc
```

```
calc: pshx ; save regs.
      psha
      ldx #point ; glbl
      ldaa 0,x
      adda 1,x
      adda 2,x
      staa sum
      pula ;rest. regs.
      pulx
      rts
```

7

Pass and return value in register

```
   ldaa num    ; get number
   jsr mul5    ; multiply by 5
   staa output ; save it

mul5: ; multiply by 5
   pshb   ; preserve registers used: B
   tab    ; copy argument to B
   aslb   ; B << 1 or B *= 2
   aslb   ; ...again
   aba    ; now *5 finished
   pulb   ; restore preserved registers
   rts
```

8

Pass by reference in register

```
   ldx #str
   jsr toupper

; Convert C string to
; uppercase
; arg: pointer to string
; returns: nothing
toupper:
   pshx      ; preserve IX
L1:ldaa 0,x  ; get char
   beq L3    ; if null stop
   cmpa #'a  ; lowercase?
   blo L2
   cmpa #'z
   bhi L2
   adda #'A-'a ; make upper
   staa 0,x
L2:inx       ; get next char
   bra L1    ; check next
L3:pulx      ; restore regs.
   rts
```

9

What do compilers do?

- Stack and/or registers?
  - Combination?
- Order of arguments?
- What does our GCC 3.3.5 compiler do, in particular?

10

Using C++ for Embedded Systems

- Entire program in C++...
- ...or mix C++ with assembly. Assembly used for...
  - Critical code (size and efficiency?)
  - Accessing certain hardware
  - Existing, proven code (*e.g.*, reusing your matrix keypad code)

11

Mixing C++ & ASM

- Concept: Soft registers
  - Simulate additional registers by using RAM
  - Used by compiler when chip doesn't have enough "real" hardware registers
  - On page 0 – allows direct mode.
- Method: C++ functions in ASM
  - Prologs/epilogs – function saves and restores registers, if necessary, for caller
  - Accessing parameters from stack...
- Method: Inline assembly...
- Method: `extern` variables...

12

C++ Functions in ASM – Accessing Parameters

- General procedure
  - Load an index register with the correct stack frame
  - Used indexed mode loads and stores to read/write
- Details: http://people.msoe.edu/~durant/courses/cs280/passing.shtml

13

C++ Functions in ASM – Parameters Example

```
; byte mean(byte x1, byte x2);

       .section .text
       .global mean

mean:   tsx      ; prolog
                 ; SP+1->IX (last used stack location)
                 ; [0:1],IX (return PC)
                 ; [2+],IX -> parameters after first
                 ; the 1st 8-bit parameter is in B
        addb 3,x ; add the 2nd 8-bit parameter
        lsrb     ; unsigned division by 2 (truncate)
                 ; 8-bit return value is in B
        rts      ; epilog
```

14

Inline Assembly

- Useful when writing mostly in C++ but a few ASM instructions are needed for...
  - Calling ASM subroutines
  - Accessing ASM global variables
    - (Not covered in detail)
  - Maximum efficiency for a small but often used piece of code

15

Inline ASM – calling function

```
void main()
{
    ...
    __asm("jsr setUpHardware");
    // Call routine without
    // a C++ interface
    ...
}
```

16

`extern` variables

- **`extern`** – The definition and label for something...
  - are external (present in another module)
  - will be resolved by the linker
- **`extern`** modifies a declaration
- Examples
  - Variables – stored in another module
  - Functions – implemented in another module

17

Access ASM Variable from C++

- **Assembly**

```
          .section .data
          .global counter
counter: .word 0x1234
```

- **C++**

```
int main()
{
    extern volatile word counter;
    myLCDDipslay.outputNumber(counter, 5);
}
```

18