Course Administration

NEW LAB SUBMISSION POLICY (by Joel Wilder)

- Turn in a printout of your code to the instructor personally, or slide it under his office door. If the instructor has to print out your file, you will have 10 points subtracted from your grade.
- Demonstrate the code for the instructor in the lab hour. If this is not possible before the deadline, email the code to the instructor by the due date.
- Assignments are due by 8pm on the due date (assuming this is on a Wednesday. Make it by 3pm if this is a Friday).

The Stack and Local Variables

- Subroutines often need local workspace
  - We can use a fixed block of memory space – static allocation – but:
    - The code will not be relocatable
    - The code will not be reentrant
    - The code will not be able to be called recursively
  - Better solution: dynamic allocation
    - Allocate all local variables on the stack
    - STACK FRAME = a block of memory allocated by a subroutine to be used for local variables
    - FRAME POINTER = an address register used to point to the stack frame

The Stack and Local Variables

- LINK and UNLK automate the creation and removal of the stack frame
  - LINK (P) set up SP as frame pointer
  - UNLK (P) remove the stack frame

Implementation

```
LINK (P) \n(SP) => [SP] - 4 \n(SP) => SP \n
UNLK (P) \n(SP) => [SP] + 4 \n(SP) => SP + 4
```
The Stack and Local Variables

Nested subroutines: A calls B, then B calls A

PEA Char Push address of dest. for the input
PEA Error_Status Push address of Error_Status message
PEA ACIA Push ACIA's address on the stack
MOVE.W Function,-(A7) Push value of function code on the stack
LEA Char_In Call subroutine
LEA (14,A7),A7 Clean up the stack - remove the four parameters

Character_Input and ACIA_Initialize routine
* SF location A6 - 6 holds the ACIA's status
* SF location A6 - 4 holds the ACIA's masked status (error bits only)
* SF location A6 - 2 holds the Cycle_Count
* A1 contains the address of the Error_Status
* A2 contains the address of the ACIA's control/status register

Char_In LINK A6,#-6 Create a
MOVEM.L A1-A2,-(A7) Push working registers on the stack
MOVEA.L (14,A6),A1 Read address of Error_Status from
the stack
MOVEA.L (10,A6),A2 Read address of ACIA
CLR.B (A1) Clear Error_Status
MOVE.W #$FFFF,(-2,A6) Set up Cycle_Count for timeout
CMPI.B #0,(8,A6) IF Function not zero THEN get input
BNE InPut ELSE initialize ACIA
MOVE.B #3,(A2) Reset ACIA
MOVE.B #$19,(A2) Configure ACIA
BRA Exit_2 Return after initialization

InPut MOVE.B (A2),(-4,A6) Read the ACIA's status register -
save in Temp1
MOVE.L (A6),A1 Get address for data dest.
* (reuse A1)
MOVE.B (2,A2), (A1) Read data from ACIA
BRA Exit_2

Exit_1 MOVE.B (-6,A6),(A1) Return Error_Status
Exit_2 MOVEM.L (A7)+,A1-A2 Restore working registers

C and The 68000

- Compiler and 68000 instruction set
- C data types and implementation
- Storage classes
- Functions and parameters
- Pointers
Compiling a C Program

```c
void main (void) {
    int i;
    int j;
    i = 1;
    j = 2;
    i = i + j;
}
```

/* Comments
SECTION S_main,"code"
XREF __main
*XDEF _main
_main
LINK A6,#-4

* Variable i is at -2(A6)
*XREF _main
_unlk A6
RTS
```

C Data Types

- The 68000 family supports three basic data types:
  - Bytes, Word, Longword
  - Each can be interpreted as signed or unsigned
- C built-in types:
  - Integer, character, floating point, double-precision
  - Void – refers to the null data type
  - Implementation dependant!

<table>
<thead>
<tr>
<th>Data type</th>
<th>C name</th>
<th>Width (b)</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>int</td>
<td>16</td>
<td>-32768 to 32767</td>
</tr>
<tr>
<td>short integer</td>
<td>short</td>
<td>8</td>
<td>-128 to 127</td>
</tr>
<tr>
<td>long integer</td>
<td>long int</td>
<td>32</td>
<td>-2147483648 to 2147483647</td>
</tr>
<tr>
<td>unsigned integer</td>
<td>unsigned int</td>
<td>16</td>
<td>0 to 65535</td>
</tr>
<tr>
<td>character</td>
<td>char</td>
<td>8</td>
<td>0 to 255</td>
</tr>
<tr>
<td>single-precision floating point</td>
<td>float</td>
<td>32</td>
<td>10^-38 to 10^38</td>
</tr>
<tr>
<td>double-precision floating point</td>
<td>double</td>
<td>64</td>
<td>10^-300 to 10^300</td>
</tr>
</tbody>
</table>

C Data Types, cont’d

- Local variables
  - Defined inside a function
  - Cannot be accessed from outside the function
  - Normally lost when a return from the function is made
- Global variables
  - Defined outside a function
  - Can be accessed both from inside and outside the function
- Variables defined in a block exist only within that block

```c
int i; /* global variable, visible to everything from this point */
void function_1 (void) /* A function with no parameters */
{
    int k; /* Integer k is local to function_1 */
    int j; /* Integer j is local and not the same as j in main */
}
void main (void)
{
    int j; /* Integer j is local to this block within function main */
    /* This is the point at which integer j ceases to exist */
}
```

Storage Class

- Storage class specifiers
  - auto
    - Variable is no longer required once a block has been left; Default
  - register
    - Ask compiler to allocate the variable to a register
    - Also is automatic
    - Cannot be accessed by means of pointers
  - static
    - Allows local variable to retain its value when a block is reentered
    - Initialized only once, by the compiler!
  - extern
    - Indicates that the variable is defined outside the block
    - The same global variable can be defined in more than one modul

```c
volatile
• To define variables that can be changed externally
• Compiler will not put them in registers
• Think about Status Registers!

const
• Variable may not be changed during the execution of a program
• Cannot be changed unintentionally, but CAN be changed externally
  (as a result of an I/O, or OS operations external to the C program)
```

Type conversion

- In C, done either automatically or explicitly (casting)
  ```c
  int x; /* Global variable, visible to everything from this point? */
  void function_1(void) /* A function with no parameters */
  {
      int x; /* Integer x is local to function_1 */
      int y; /* Integer y exists only in this block. */
      int z; /* Integer z is local and not the same as z in main */
  }
  void main(void)
  {
      int j; /* Integer j is local to this block within function main */
      /* This is the point at which integer j ceases to exist */
  }
  ```

Returning a Value from a Function

- Example: main calls function adder
  ```c
  int adder(int a, int y); /* returns an integer */
  return a + y; /* return sum of a and y to the calling program */
  void main(void)
  {
      int a, b, c; /* assign variables a, b, and c to regs */
      a = b + 2; /* provide some dummy values for a and b */
      c = adder(a, b); /* c is assigned the integer returned by adder */
  }
  ```

Access Modifiers

- volatile
  - To define variables that can be changed externally
  - Compiler will not put them in registers
  - Think about Status Registers!
- const
  - Variable may not be changed during the execution of a program
  - Cannot be changed unintentionally, but CAN be changed externally
    (as a result of an I/O, or OS operations external to the C program)

Storage Class, cont’d

- volatile
  - To define variables that can be changed externally
  - Compiler will not put them in registers
  - Think about Status Registers!
- const
  - Variable may not be changed during the execution of a program
  - Cannot be changed unintentionally, but CAN be changed externally
    (as a result of an I/O, or OS operations external to the C program)
CPE/EE 421/521 Microcomputers

Returning a Value from a Function, cont'd

```c
* Parameter x is at 8(A6)
* Parameter y is at 10(A6)

int adder(int x, int y)
{
    int a, b, c;
    a = 1, b = 2;
    c = adder(a, b);
    return x + y;
}
```

Arrays

```c
void main(void)
{
    int a = -2(A6);
    Variable x is at -2(A6)
    x = 1, y = 2;
    MOVE 8(A6),D1
    ADD 10(A6),D1
    MOVE D1,D0
    UNLK A6
    RTS
}
```

Pointers and C

- C is pointer-oriented
- Pointers in 68000 assembly language

```c
x=y; MOVE (AO),DO x is in DO, y is in AO
a=b; LEA B,A0 a is in AO
```

Pointers and C, cont'd

```c
void main(void)
{
    int x;
    Variable x is at -2(A6)
    P_port = int * (int*) 0x4000;
    Variable P_port is at -6(A6)
    P_port = int * (int*) 0x4000;
    while (*P_port & 0x0001 == 0) {
        x = * (P_port + 1);
    }
    UNLK A6
    RTS
}
```
Passing parameters to a function
- Passing by value/reference
- Is this going to work?

/* this function swaps the values of a and b */
void swap (int a, int b) {
    int temp; /* copy a to temp, b to a, and temp to b */
    temp = a;
    a = b;
    b = temp;
}
void main (void) {
    int x = 2, y = 3;
    swap(x, y); /* let’s swap a and b */
}

No, because this program is using a call-by-value mechanism

To permit the function to modify the parameters, pass the address of the parameters
This will work...
/* swap two parameters in the calling program */
void swap (int *a, int *b) {
    int temp; /* a = *b; */
    *a = *b;
    *b = temp;
}
void main (void) {
    int x = 2, y = 3;
    swap(&x, &y); /* let’s swap a and b */
}
Functions and Parameters
USE OF STACK, Call-by-reference

- State of the stack after
  - RP Already set
  - Move parameter
do not change
  - Saving return address

- State of the stack after
  - Parameter address
  - Return address
  - Temporary

- State of the stack after
  - Parameter address
  - Temporary

Figure 3.12

Speed and Performance of Microprocessors

- Why is difficult to compare the speed of two microprocessors?
  1. Clock speed
  2. Meaningless MIPS
  3. Memory access times
  4. Are registers used optimally?
  5. Special addressing modes (not generally useful)
  6. Misleading benchmarks
  7. Use of cache
  8. Pipeline

- Carefully interpret benchmarks!
- Clock Cycles/Bus Cycles

Example: Interpret the high-level language construct

IF COUNT[CLASS(11)] <> 0 THEN …

68000 Version

<table>
<thead>
<tr>
<th>Clock</th>
<th>Bus</th>
<th>Clock</th>
<th>Bus</th>
<th>Cycles</th>
<th>Cycles</th>
<th>Cycles</th>
<th>Cycles</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>MOVE.W D1,D3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>LSL.W $1,D3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>6</td>
<td>0</td>
<td>LEA 0(A5,D3,W),A2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>3</td>
<td>7</td>
<td>1</td>
<td>MOVE.W CLASS(A2),D3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>LSL.W $1,D3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>6</td>
<td>0</td>
<td>LEA 0(A5,D3,W),A2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>3</td>
<td>7</td>
<td>1</td>
<td>TST.W COUNT(A2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>6</td>
<td>0</td>
<td>BRQ ELSE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>79</td>
<td>15</td>
<td>42</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

68020 Version

<table>
<thead>
<tr>
<th>Clock</th>
<th>Bus</th>
<th>Clock</th>
<th>Bus</th>
<th>Cycles</th>
<th>Cycles</th>
<th>Cycles</th>
<th>Cycles</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>MOVE.W D1,D3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>LSL.W $1,D3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>6</td>
<td>0</td>
<td>LEA 0(A5,D3,W),A2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>3</td>
<td>7</td>
<td>1</td>
<td>MOVE.W CLASS(A2),D3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>LSL.W $1,D3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>6</td>
<td>0</td>
<td>LEA 0(A5,D3,W),A2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>3</td>
<td>7</td>
<td>1</td>
<td>TST.W COUNT(A2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>6</td>
<td>0</td>
<td>BRQ ELSE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>79</td>
<td>15</td>
<td>42</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

MIPS = Million Instructions Per Second

- For the previous example, 68000:
  - Execution time = 6.32 µs
  - => 8 instructions / 6.32 µs = 1.27 MIPS

- 68020 using the same code
  - Execution time = 2.52 µs
  - => 8 instructions / 2.52 µs = 3.17 MIPS

- 68020 using special features
  - Execution time = 1.44 µs
  - => 3 instructions / 1.44 µs = 2.08 MIPS

Example

For the given assembly language program:

LEA TABLE,A0
CLR.W D1
LOOP MOVE.B D0,(A0)+
ADDC.W $1,D1
CMP.W $9,D1
BNE LOOP

- Find the total execution time of the given program on a 12.5 MHz 68000 microprocessor.
- What is the average CPI (number of clocks per instructions)?
- What is the MIPS rate?
b) What is the average CPI (number of clocks per instructions)?

Number of clocks/instruction

<table>
<thead>
<tr>
<th>Number of clock cycles to execute the program</th>
<th>Total number of instructions in the program (loops!)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CPI = ( \frac{C}{N} = \frac{280}{38} = 7.37 )</td>
</tr>
</tbody>
</table>

b) What is the MIPS rate?

\[
\text{MIPS rate} = 10^6 \times \frac{f}{\text{CPI}} = 12.5 / 7.37 = 1.7 \text{ MIPS}
\]