Midterm Review

• 1. Branch instructions
  ▪ BHI (unsigned), BGT (signed)
    • Take a look at the preceding comparison instruction.
    • Then, you can use this instead of using complex formula in the instruction reference.
      ▪ **BRANCH IF REGISTER IS HIGHER/GREATHER/... THAN OPERAND**
    • e.g. CMPA #$D0
      ▪ Branch if ‘A’ is Higher than ‘operand.’
  ▪ BMI, BVC, BNE
    • Check the result if it is Minus/signed oVerflow/Not Equal to zero
  ▪ CMPA: A – (M)
  ▪ TSTA: A – 0
  ▪ V bit (signed overflow)
    • N + N → P
    • P + P → N
    • N – P → P
    • P – N → N
2. BRA and LBRA

- Write a line of assembly code that begins in memory location $2450.
- BRA <8-bit signed offset value>
- LBRA <16-bit signed offset value>
- When the effective (destination) address is calculated in relative addressing mode, a value that you need to add to an offset is PC.
  → We have to use PC which is the next address of the line of assembly code of (L)BRA
Midterm Review

3. Mostly this is about calculating postbyte(s)
   -20: You cannot use 5-bit offset. You have to use 9-bit one.
Lecture 16: Parameter Passing
Today’s Goals

- Parameter passing
- Understand how to pass parameters using the stack
Call by Value vs. Call by Reference

Diagram:
- Main Program
- Subroutine
- Subroutine
- Memory

Flow:
- Main Program sends Value to Subroutine.
- Main Program sends Address to Subroutine.
- Subroutine looks for values in Memory.
Passing Parameters in Registers

- Use registers to pass parameters
- The simplest type of parameter passing
- Pros vs. Cons
  - Data is immediately available to the subroutine
  - No extra preparations are needed.
  - Often fastest execution and smallest code size
  - Only a limited number of registers are available.
  - We can use ‘pass–by–reference’ to point a list of input/output values.
Example (Parameter Passing in Registers)

• Requirements
  ▪ A subroutine that adds an array of two-byte numbers and a sample main program that calls it.
  ▪ The array is passed by reference in X.
  ▪ The length is passed by value in D.
  ▪ The sum should be returned by value in D.
  ▪ Do not worry about indicating signed or unsigned overflow.
Example (Parameter Passing in Registers)

```
ORG $3000
Array dc.w $1234,$5678,$ABCD
Length dc.w 3

ORG $2000
LDS #$3600
LDX #Array ; load X with address of list
LDD Length ; Load D with actual length
JSR sumword
SWI

sumword TFR D,Y
LDDD #0
CPY #0
Loop BEQ endsum
ADDD 0,X
INX
INX
DEY
BRA loop
endsum RTS
```
Preserving Registers

• In the previous example, the subroutine uses Y. The value in Y is destroyed.

• If Y was in use by the main program, the main program would continue after the subroutine with an incorrect value in Y.

• To avoid this, registers used by a subroutine may be saved to the stack before a subroutine uses them.

• The saved registers are restored after the subroutine.

• We have two options
  ▪ The caller (main program) does this.
  ▪ The callee (subroutine) does this.
Preserving Registers

- **Responsibility of the Caller:**
  - The calling main program assumes all registers are destroyed by the subroutine
  - If the registers used by the subroutine are unknown (i.e. using a sub. provided by someone else), may save registers that the subroutine wouldn’t affect
  - Code to save/restore registers duplicated with every subroutine call.

- **Responsibility of the Callee:**
  - Saves only the registers that will be used by the subroutine
  - Save/restore code only occurs once
Example – Caller Responsible

Example of a Caller Responsible Routine:

```
ORG $3000
Array dc.w $1234,$5678,$ABCD
Length dc.w 3

ORG $2000
LDS #$3600
LDX #Array ; load X with address of list
LDD Length ; Load D with actual length
PSHX
JSR sumword
PULX
SWI

sumword TFR D,Y
LDD #0
CPY #0
loop BEQ endsum
ADDD 0,X
INX
INX
DEY
BRA loop
endsum RTS
```

Stack Frame

<table>
<thead>
<tr>
<th>RAH</th>
<th>RAL</th>
<th>XH</th>
<th>XL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example – Callee Responsible

```
ORG $3000
Array dc.w $1234,$5678,$ABCD
Length dc.w 3
ORG $2000
LDS #$3600
LDX #Array ; load X with address of list
LDD Length ; Load D with actual length
JSR sumword
SWI

sumword PSHX
PSHY
TFR D,Y
LDDD #0
CPY #0
loop BEQ endsum
ADDD 0,X
INX
INX
DEY
BRA loop
endsum PULY
PULX
RTS
```

Why not D?
It is used to return the answer!
Passing Parameters in the Stack

• Pros and Cons
  ▪ The stack pointer is already in use for the return address
  ▪ Indexed addressing can easily access data stored on the stack
  ▪ The amount of data passed is not limited by the register set
  ▪ The data passed on the stack must be removed, and this is usually the responsibility of the caller (although this can be done by the callee)

• Note:
  ▪ If the caller will save registers on the stack, it should be done before passing parameters.
  ▪ Why? Those parameters are supposed to be used in the subroutine.