**LEC 3**

* **Procedures in MIPS**
  + Problems:
  + Call & return – transferring control into/out of procedures
  + Registers – what if procedure overwrites my registers?
    - Guarantee that procedures leave registers unchanged when they’re done
    - Use RAM to store existing register values
    - Can’t let procedures use the same RAM
    - $30 is initialized by the loader to just past the last word of memory – use to determine where free RAM is located
    - $30 moves “up” as RAM is used by procedures, moves “down” as RAM is freed
    - i.e. the RAM stack
    - $30 is the stack pointer – points to the address at the top of the stack
    - Stack is full descending
* **Template for procedures:**
  + f: sw $2, -4($30)
  + sw $3, -8($30) – push registers that will be modified onto the stack
  + lis $3
  + .word 8
  + sub $30, $30, $3 – update (decrement) $30
  + …
  + add $30, $30, $3 – update (increment) $30
  + lw $3, -8($30)
  + lw $2, -4($30) – load previous register values
  + jr $31 – return to main
* **Call & return**
  + In order to return a procedure, set PC to the line after the jr
  + **jalr** **$s – jump and link register**
    - Works like jr, except sets $31 to the address of the next instruction
    - Have to save $31 on a stack before the call, then restore it afterwards
  + main: …
  + lis $5
  + .word f – address of line labelled f
  + sw $31, -4($30) – push $31 to stack
  + lis $31
  + .word 4
  + sub $30, $30, $31 – decrement $30
  + jalr $5 – jump to f, saves address of next line in $31 so f can return
  + lis $31
  + .word 4
  + add $30, $30, $31 – increment $30
  + lw $31, -4($30) – restore $31
  + jr $31
* **Parameters & results**
  + Generally use registers; use the stack if there are too many
  + i.e. load parameters into registers (in main) before calling procedure
  + ; sum1toN: computes 1 + … N
  + ; registers
  + ; $1 – working
  + ; $2 – input (value of N)
  + ; $3 – output (don’t need to save)
  + sum1toN:
  + sw $1, -4($30)
  + sw $2, -8($30)
  + lis $1
  + .word 8
  + sub $30, $30, $1 – save registers and decrement $30
  + add $3, $0, $0
  + top: add $3, $3, $2 – add sum
  + lis $1
  + .word 1
  + sub $2, $2, $1 – decrement counter
  + bne $2, $0, top – loop
  + lis $1
  + .word 8
  + add $30, $30, $1
  + lw $2, -8($30)
  + lw $1, -4($30) – recover registers and increment $30
  + jr $31 – return
* **Recursion**
  + Recursion can be accommodated without anything extra if registers, params, and stack are managed properly
* **Output**
  + Use sw to store a word in location 0xffff000c
  + The last byte in the word will be printed on the screen
  + lis $1
  + .word 0xffff000c
  + lis $2
  + .word 67 ;’C’
  + sw $2, 0($1) ;print the char
  + lis $2
  + .word 83 ;’S’
  + sw $2, 0($1) ;print the char