Anatomy of an Assembly Language Source File

CS4212 – Lecture 2 (prerecorded)

Factorial in Assembly Language

factorial.S

```
.section .text          # section for code
.globl _start           # start address accessible by linker

_start:  # entry point to the program
    movl 8(%esp),%esi # esi = addr of first cmd line arg
    xorl %eax,%eax     # eax = 0

loop1: # convert cmd arg string to int, one digit at a time
    cmpb $0,(%esi) # check for '\0'
    jz exitloop1   # if end-of-string reached, move on
    xor %edx,%edx
    movb (%esi),%dl# load next char from string into dl
    subl $'0',%edx # edx -= '0' : turn digit char into int
    imull $10,%eax # shift digits to left
    add %edx,%eax  # add another digit
    incl %esi      # increment pointer to cur char
    jmp loop1      # continue
exitloop1: # result of conversion in eax

    # compute factorial
    movl %eax,%edi # counter in edi, will be decr to 0
    movl $1,%eax   # result accumulated in rax
loop2:
    cmp $0,%edi # if end of computation reached, move on
    jz loop2exit
    imull %edi,%eax # multiply with counter
    decl %edi       # decrement counter
    jmp loop2       # continue
loop2exit:
```

```
    # convert int result into string
    movl $result+99,%esi # string pointer at end of buffer
    movb $'\n',(%esi)    # add EOL char
    movl $1,%ecx         # counter for length of string
loop3:              # produce digits in reverse order
    decl %esi       # dec ptr to cur char position in string
    xorl %edx,%edx # edx = 0
    movl $10,%ebx  # ebx = 10
    idivl %ebx      # edx = last digit of result
    addl $'0',%edx # convert int to char
    movb %dl,(%esi)# store char into string
    incl %ecx       # increase ctr reflecting length of string
    cmpl $0,%eax   # check if done
    jnz loop3       # if not, continue

    movl $4,%eax   # print string via system call
    movl $1,%ebx   # expected args
    movl %ecx,%edx # eax = 1 (write), ebx=1 (stdout)
    movl %esi,%ecx # ecx = start of string, edx = length
    int $0x80       #

    movl $0,%ebx # exit, eax = 1 (exit), ebx = exit code
    movl $1,%eax
    int $0x80

.section .bss
    .lcomm result,100 # buffer to store result
```

Factorial in Assembly Language

[B]

factorial.S

```
.section .tex
.globl _start

_start:  # en
    movl 8(%e
    xorl %eax

loop1: # conv
    cmpb $0,(
    jz exitlo
    xor %edx,
    movb (%es
    subl $'0'
    imull $10
    add %edx,
    incl %esi
    jmp loop1
exitloop1: #

    # compute
    movl %eax
    movl $1,%
loop2:
    cmp $0,%e
    jz loop2e
    imull %ed
    decl %edi
    jmp loop2
loop2exit:
```

Obtain an executable in Linux

```
$ as -ggdb -o factorial.o factorial.S
$ ld -o factorial factorial.o
$ ./factorial 5
120
```

Factorial in Assembly Language

[B]

factorial.S

```
.section .tex
.globl _start

_start:  # en
    movl 8(%e
    xorl %eax

loop1: # conv
    cmpb $0,(
    jz exitlo
    xor %edx,
    movb (%es
    subl $'0'
    imull $10
    add %edx,
    incl %esi
    jmp loop1
exitloop1: #

    # compute
    movl %eax
    movl $1,%
loop2:
    cmp $0,%e
    jz loop2e
    imull %ed
    decl %edi
    jmp loop2
loop2exit:
```

Obtain an executable in Linux

```
$ as -ggdb -o factorial.o factorial.S
$ ld -o factorial factorial.o
$ ./factorial 5
120
```

Windows executables will be discussed later...

Structure of an Assembly file

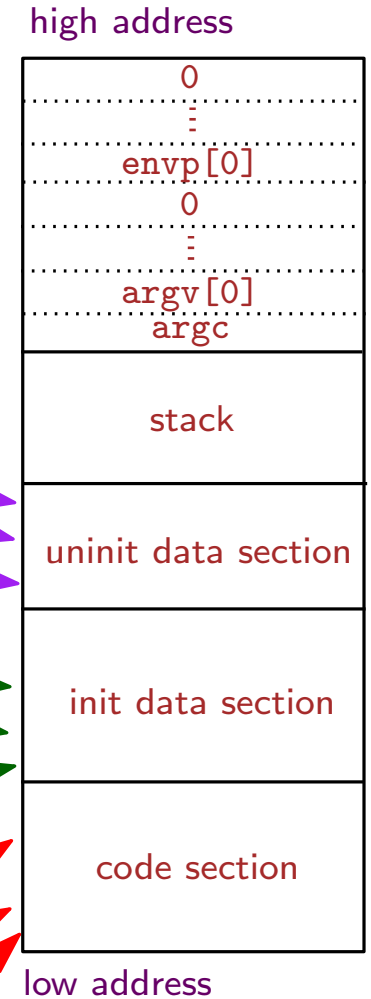
source.S

```
.section .text
   ...fragment of code1
.section .data
   ...fragment of initalized data1
.section .bss
   ...fragment of uninitalized data1
.section .text
   ...fragment of code2
.section .data
   ...fragment of initalized data2
.section .bss
   ...fragment of uninitalized data2
.section .text
   ...fragment of code3
.section .data
   ...fragment of initalized data3
.section .bss
   ...fragment of uninitalized data3
```

Add to a text fragment

```
      .global _start
_start:
```

Required by the linker as the entry point of code.



About Fragment Ordering

- Do not assume any ordering of the fragments inside a section
  - The assembler will order the fragments based on alignment constraints, so as to save space.

- One fragment should contain the main program (having a label `_start`)
  - Terminate this fragment with the OS's "exit" system call

- All other fragments should contain complete procedures
  - When the procedure is called, the entire fragment executes
  - The fragment ends with the return from the call

Demo

Explanation of the code and demonstration of debugging

Mixing AL and C

- Some operations that are routine in most high-level languages are tedious in AL
  - Converting a string into an integer
  - Converting an integer into a string
  - Printing a string
  - Exiting a program

- Certain operations are not portable across multiple OSs
  - print string and exit system services will not work in Windows

- We can write AL functions that can be called from C and viceversa

- Our approach for this module
  - Devise toy languages and compile them into an AL function
  - Use a "stock" C program wrapper that calls the AL function so as to verify that our compiler's output is correct.
  - The stock C program will abstract away OS differences and simplify input/output operations

Calling C from AL

fact1.S

```
.section .text
.globl fact

fact:
    pushl   %ebp        # prologue
    movl    %esp, %ebp
    movl    8(%ebp),%eax # load argument

    # compute factorial
    movl %eax,%edi # counter in edi, will be decr to 0
    movl $1,%eax   # result accumulated in eax
loop2:
    cmp $0,%edi # if end of computation reached, move on
    jz loop2exit
    imull %edi,%eax # multiply with counter
    decl %edi       # decrement counter
    jmp loop2       # continue
loop2exit:

    movl    %ebp,%esp   # epilogue, eax=return reg
    popl    %ebp
    ret
```

fact2.c

```
#include <stdio.h>

// asm("fact") ensures compatibility
//             between Win and Linux
int fact(int) asm("fact") ;

int main() {
  printf("factorial of 5: %d\n",fact(5));
}
```

Compiling and running

Linux:

```
$ gcc -m32 fact1.S fact2.c -o fact
$ ./fact
120
```

Windows (Cygwin or MinGW):

```
> gcc -m32 fact1.S fact2.c -o fact.exe
> fact
120
```