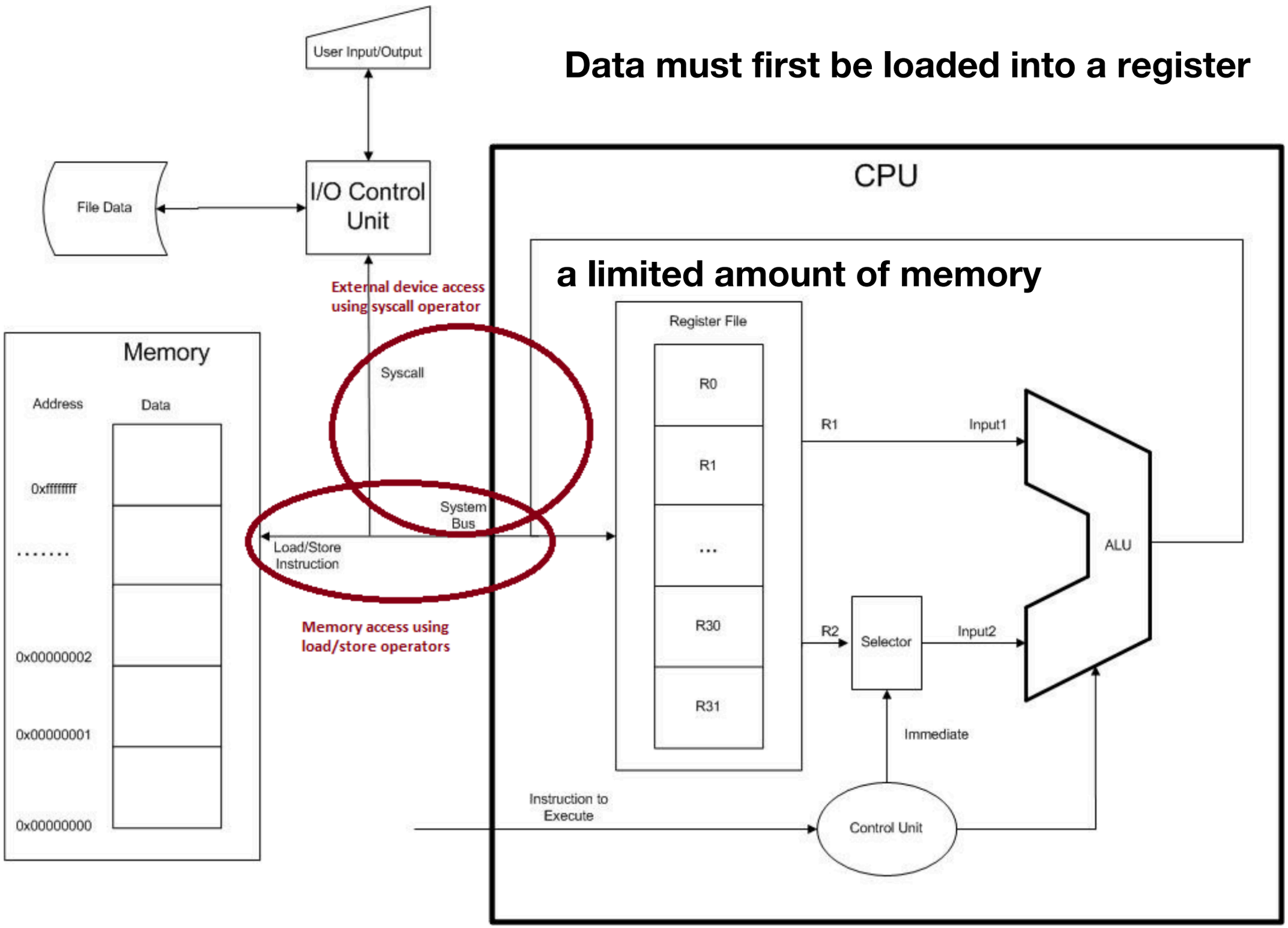
Assembly Language and Computer Architecture Lab

Nguyen Thi Thanh Nga
Dept. Computer engineering
School of Information and Communication Technology

Week 2

- Registers and memory in MIPS computers
- How to comment a MIPS program
- Assembler directives such as **.text**, **.data**, **.asciiz**, **.space**, and **.word**
- Labels in MIPS assembly
- The MIPS assembly operators such as **li**, **la**, **lw** and **move**
- System services for interacting with the user console, in particular services 1, 4, 5, and 8.
- The difference between references and values of data.

MIPS CPU Architecture



Registers

$zero ($0):

- A special purpose register
- Always contains a constant value of 0.
- Can be read, but cannot be written.

Mnemonic	Number	.....	Mnemonic	Number	.....	Mnemonic	Number
$zero	$0		$t3	$11		$s6	$22
$at	$1		$t4	$12		$s7	$23
$v0	$2		$t5	$13		$t8	$24
$v1	$3		$t6	$14		$t9	$25
$a0	$4		$t7	$15		$k0	$26
$a1	$5		$s0	$16		$k1	$27
$a2	$6		$s1	$17		$gp	$28
$a3	$7		$s2	$18		$sp	$29
$t0	$8		$s3	$19		$fp	$30
$t1	$9		$s4	$20		$ra	$31
$t2	$10		$s5	$21			

Registers

$at ($1):

- Reserved for the assembler
- Unavailable for the programmer

Mnemonic	Number	.....	Mnemonic	Number	.....	Mnemonic	Number
$zero	$0		$t3	$11		$s6	$22
$at	$1		$t4	$12		$s7	$23
$v0	$2		$t5	$13		$t8	$24
$v1	$3		$t6	$14		$t9	$25
$a0	$4		$t7	$15		$k0	$26
$a1	$5		$s0	$16		$k1	$27
$a2	$6		$s1	$17		$gp	$28
$a3	$7		$s2	$18		$sp	$29
$t0	$8		$s3	$19		$fp	$30
$t1	$9		$s4	$20		$ra	$31
$t2	$10		$s5	$21			

Registers

$v0-$v1 ($2-$3)

- Normally used for return values for subprograms
- $v0 is also used to input the requested service to syscall.

Mnemonic	Number	.....	Mnemonic	Number	.....	Mnemonic	Number
$zero	$0		$t3	$11		$s6	$22
$at	$1		$t4	$12		$s7	$23
$v0	$2		$t5	$13		$t8	$24
$v1	$3		$t6	$14		$t9	$25
$a0	$4		$t7	$15		$k0	$26
$a1	$5		$s0	$16		$k1	$27
$a2	$6		$s1	$17		$gp	$28
$a3	$7		$s2	$18		$sp	$29
$t0	$8		$s3	$19		$fp	$30
$t1	$9		$s4	$20		$ra	$31
$t2	$10		$s5	$21			

Registers

$a0-$a3 ($4-$7)

- Used to pass arguments (or parameters) into subprograms.

Mnemonic	Number	.....	Mnemonic	Number	.....	Mnemonic	Number
$zero	$0		$t3	$11		$s6	$22
$at	$1		$t4	$12		$s7	$23
$v0	$2		$t5	$13		$t8	$24
$v1	$3		$t6	$14		$t9	$25
$a0	$4		$t7	$15		$k0	$26
$a1	$5		$s0	$16		$k1	$27
$a2	$6		$s1	$17		$gp	$28
$a3	$7		$s2	$18		$sp	$29
$t0	$8		$s3	$19		$fp	$30
$t1	$9		$s4	$20		$ra	$31
$t2	$10		$s5	$21			

Registers

$t0-$t9 ($8-$15, $24-$25)

- Used to store temporary variables.
- The values of temporary variables can change when a subprogram is called.

Mnemonic	Number	.....	Mnemonic	Number	.....	Mnemonic	Number
$zero	$0		$t3	$11		$s6	$22
$at	$1		$t4	$12		$s7	$23
$v0	$2		$t5	$13		$t8	$24
$v1	$3		$t6	$14		$t9	$25
$a0	$4		$t7	$15		$k0	$26
$a1	$5		$s0	$16		$k1	$27
$a2	$6		$s1	$17		$gp	$28
$a3	$7		$s2	$18		$sp	$29
$t0	$8		$s3	$19		$fp	$30
$t1	$9		$s4	$20		$ra	$31
$t2	$10		$s5	$21			

Registers

$s0-$s7 ($16-$23)

- Used to store saved values
- The values of these registers are maintained across subprogram calls.

Mnemonic	Number	.....	Mnemonic	Number	.....	Mnemonic	Number
$zero	$0		$t3	$11		$s6	$22
$at	$1		$t4	$12		$s7	$23
$v0	$2		$t5	$13		$t8	$24
$v1	$3		$t6	$14		$t9	$25
$a0	$4		$t7	$15		$k0	$26
$a1	$5		$s0	$16		$k1	$27
$a2	$6		$s1	$17		$gp	$28
$a3	$7		$s2	$18		$sp	$29
$t0	$8		$s3	$19		$fp	$30
$t1	$9		$s4	$20		$ra	$31
$t2	$10		$s5	$21			

Registers

$k0-$k1 ($26-$27)

- Used by the operating system
- Not available for use programmer use.

Mnemonic	Number	.....	Mnemonic	Number	.....	Mnemonic	Number
$zero	$0		$t3	$11		$s6	$22
$at	$1		$t4	$12		$s7	$23
$v0	$2		$t5	$13		$t8	$24
$v1	$3		$t6	$14		$t9	$25
$a0	$4		$t7	$15		$k0	$26
$a1	$5		$s0	$16		$k1	$27
$a2	$6		$s1	$17		$gp	$28
$a3	$7		$s2	$18		$sp	$29
$t0	$8		$s3	$19		$fp	$30
$t1	$9		$s4	$20		$ra	$31
$t2	$10		$s5	$21			

Registers

$gp ($28)

- Pointer to global memory.
- Used with heap allocations.

Mnemonic	Number	.....	Mnemonic	Number	.....	Mnemonic	Number
$zero	$0		$t3	$11		$s6	$22
$at	$1		$t4	$12		$s7	$23
$v0	$2		$t5	$13		$t8	$24
$v1	$3		$t6	$14		$t9	$25
$a0	$4		$t7	$15		$k0	$26
$a1	$5		$s0	$16		$k1	$27
$a2	$6		$s1	$17		$gp	$28
$a3	$7		$s2	$18		$sp	$29
$t0	$8		$s3	$19		$fp	$30
$t1	$9		$s4	$20		$ra	$31
$t2	$10		$s5	$21			

Registers

$sp ($29):

- Stack pointer
- Used to keep track of the beginning of the data for this method in the stack.

Mnemonic	Number	.....	Mnemonic	Number	.....	Mnemonic	Number
$zero	$0		$t3	$11		$s6	$22
$at	$1		$t4	$12		$s7	$23
$v0	$2		$t5	$13		$t8	$24
$v1	$3		$t6	$14		$t9	$25
$a0	$4		$t7	$15		$k0	$26
$a1	$5		$s0	$16		$k1	$27
$a2	$6		$s1	$17		$gp	$28
$a3	$7		$s2	$18		$sp	$29
$t0	$8		$s3	$19		$fp	$30
$t1	$9		$s4	$20		$ra	$31
$t2	$10		$s5	$21			

Registers

$fp ($30)

- Frame pointer
- Used with the $sp for maintaining information about the stack

Mnemonic	Number	.....	Mnemonic	Number	.....	Mnemonic	Number
$zero	$0		$t3	$11		$s6	$22
$at	$1		$t4	$12		$s7	$23
$v0	$2		$t5	$13		$t8	$24
$v1	$3		$t6	$14		$t9	$25
$a0	$4		$t7	$15		$k0	$26
$a1	$5		$s0	$16		$k1	$27
$a2	$6		$s1	$17		$gp	$28
$a3	$7		$s2	$18		$sp	$29
$t0	$8		$s3	$19		$fp	$30
$t1	$9		$s4	$20		$ra	$31
$t2	$10		$s5	$21			

Registers

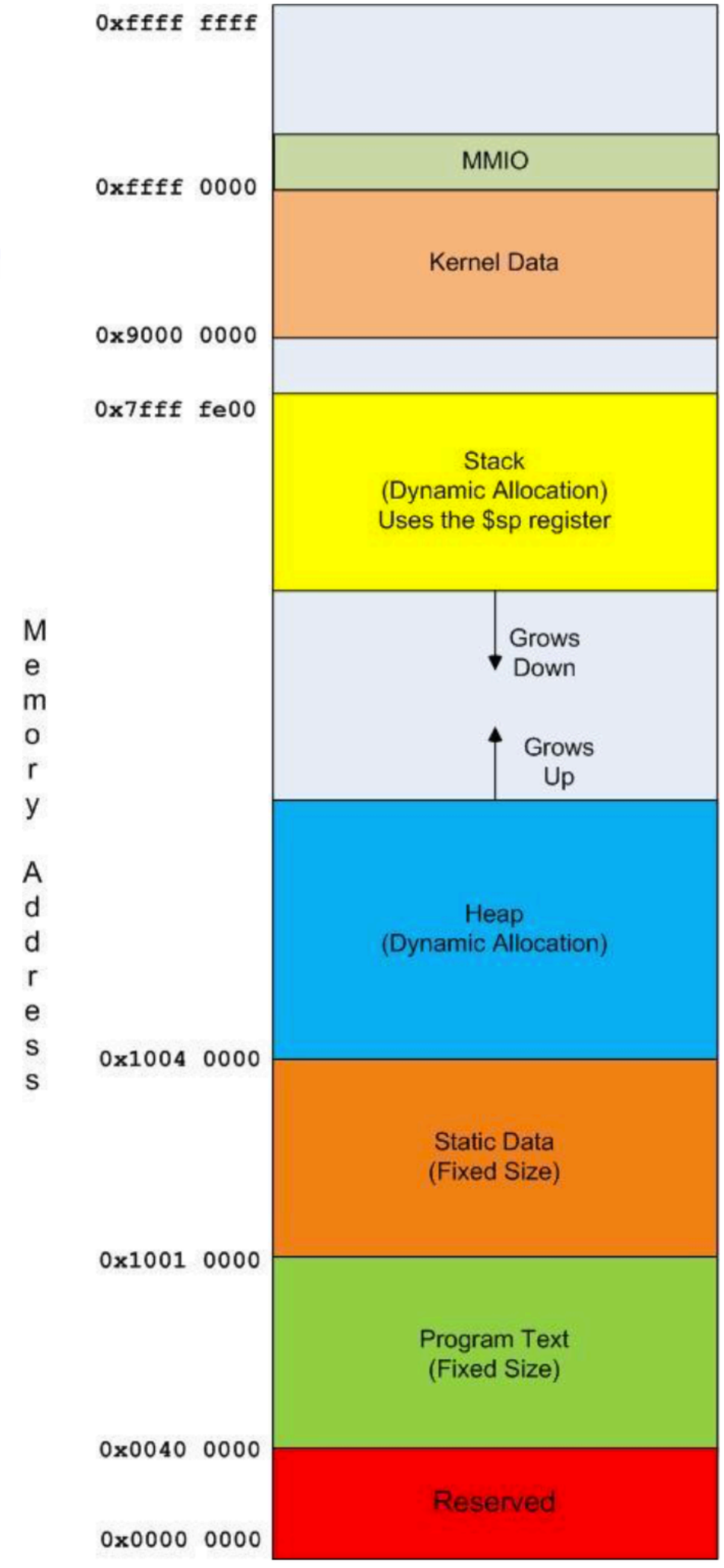
$ra ($31):

- Return address
- A pointer to the address to use when returning from a subprogram.

Mnemonic	Number	.....	Mnemonic	Number	.....	Mnemonic	Number
$zero	$0		$t3	$11		$s6	$22
$at	$1		$t4	$12		$s7	$23
$v0	$2		$t5	$13		$t8	$24
$v1	$3		$t6	$14		$t9	$25
$a0	$4		$t7	$15		$k0	$26
$a1	$5		$s0	$16		$k1	$27
$a2	$6		$s1	$17		$gp	$28
$a3	$7		$s2	$18		$sp	$29
$t0	$8		$s3	$19		$fp	$30
$t1	$9		$s4	$20		$ra	$31
$t2	$10		$s5	$21			

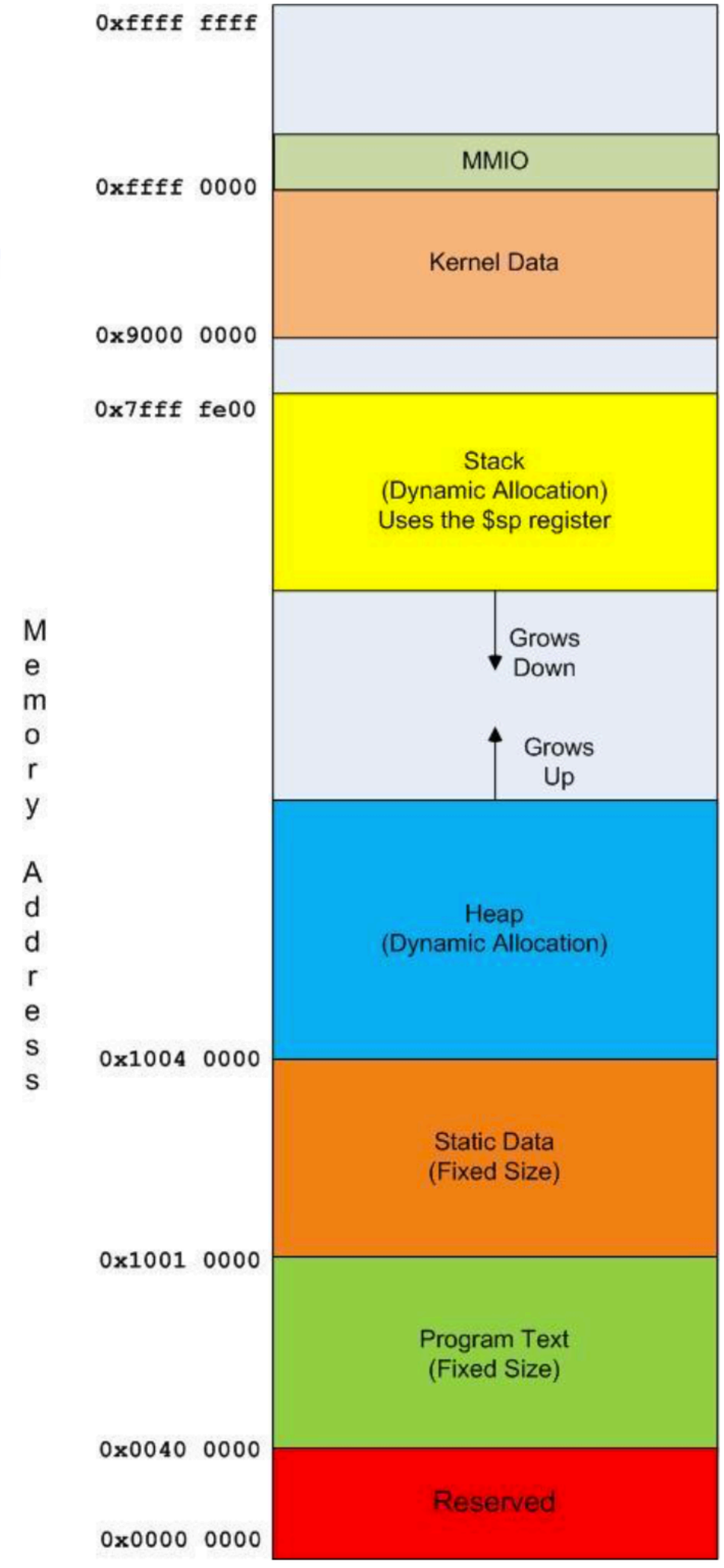
Types of memory

- 32-bit flat memory model.
- Can *address* (or find) 4 Gigabytes (4G) of data
- Starts at address 0x00000000 and extends in sequential, contiguous order to address 0xffffffff
- This does not mean that all of memory is available to the programmer.



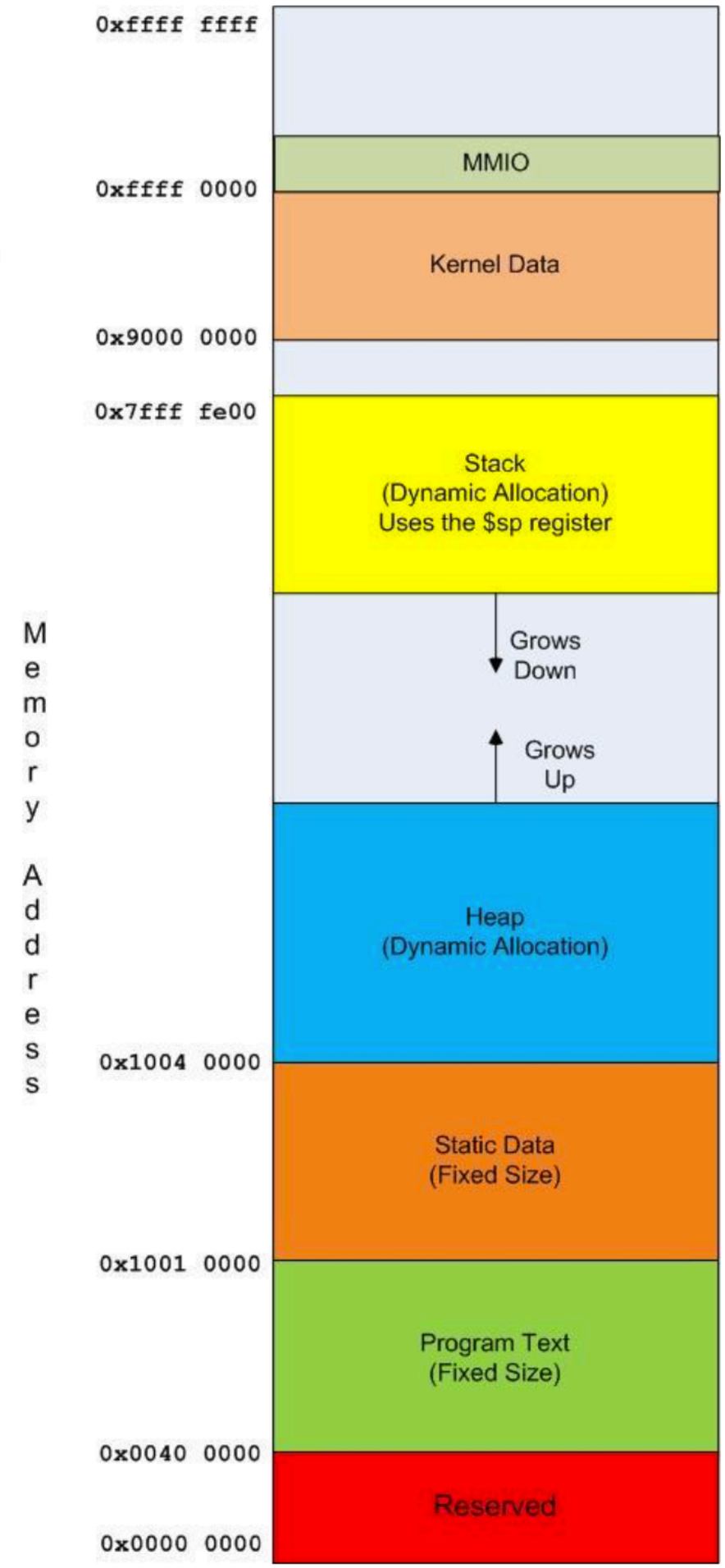
Types of memory

- Reversed: (Addresses 0x0000 0000 - 0x003f fffc)
- Is reserved for the MIPS platform.
- Memory at these addresses is not useable by a program.



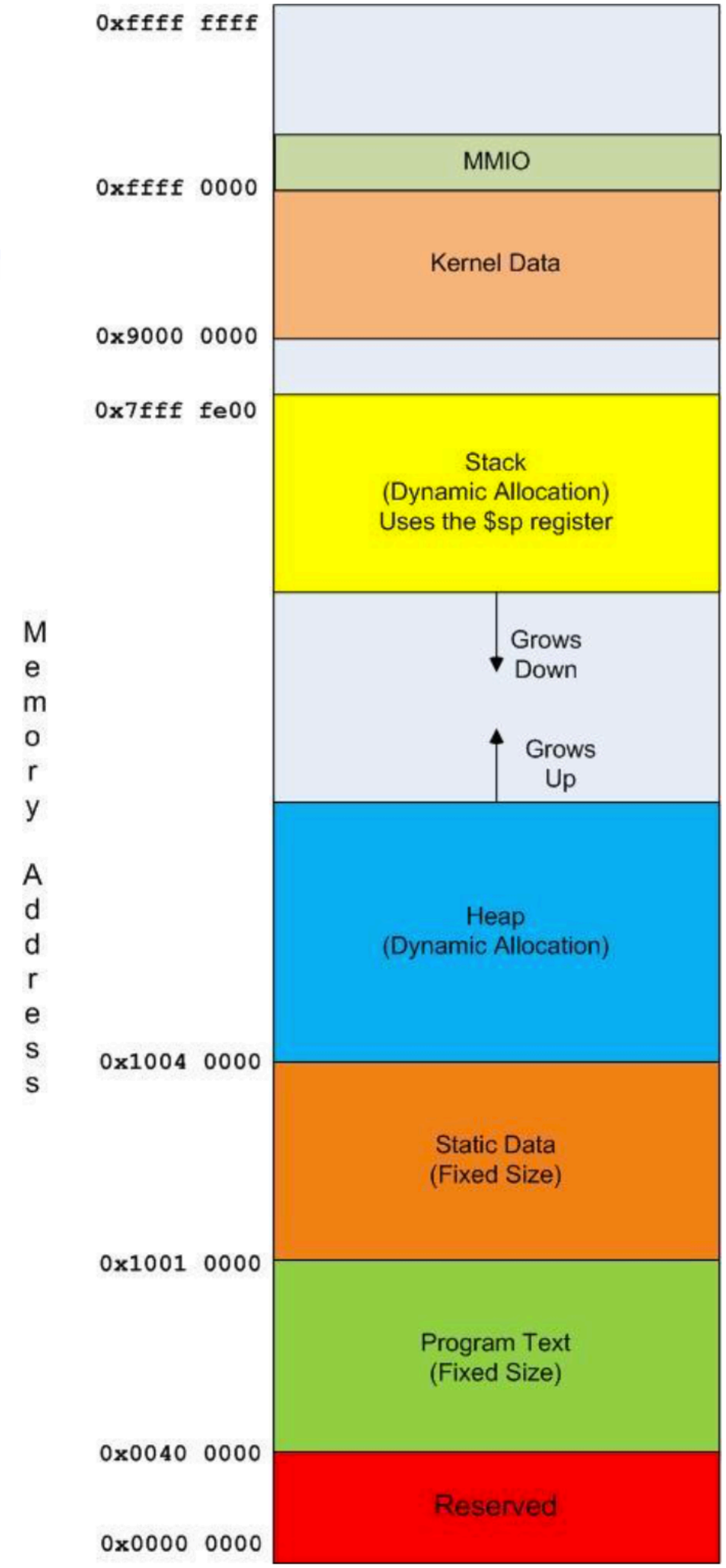
Types of memory

- Program text: (Addresses 0x0040 0000 - 0x1000 0000)
- Stores the machine code representation of the program.
- Each instruction is stored as a *word* (32 bits or 4 byte) in this memory.
- All instructions fall on a *word boundary*, which is a multiple of 4 (0x0040 0000, 0x0040 0004, 0x0040 0080, 0x0040 00B0, etc).



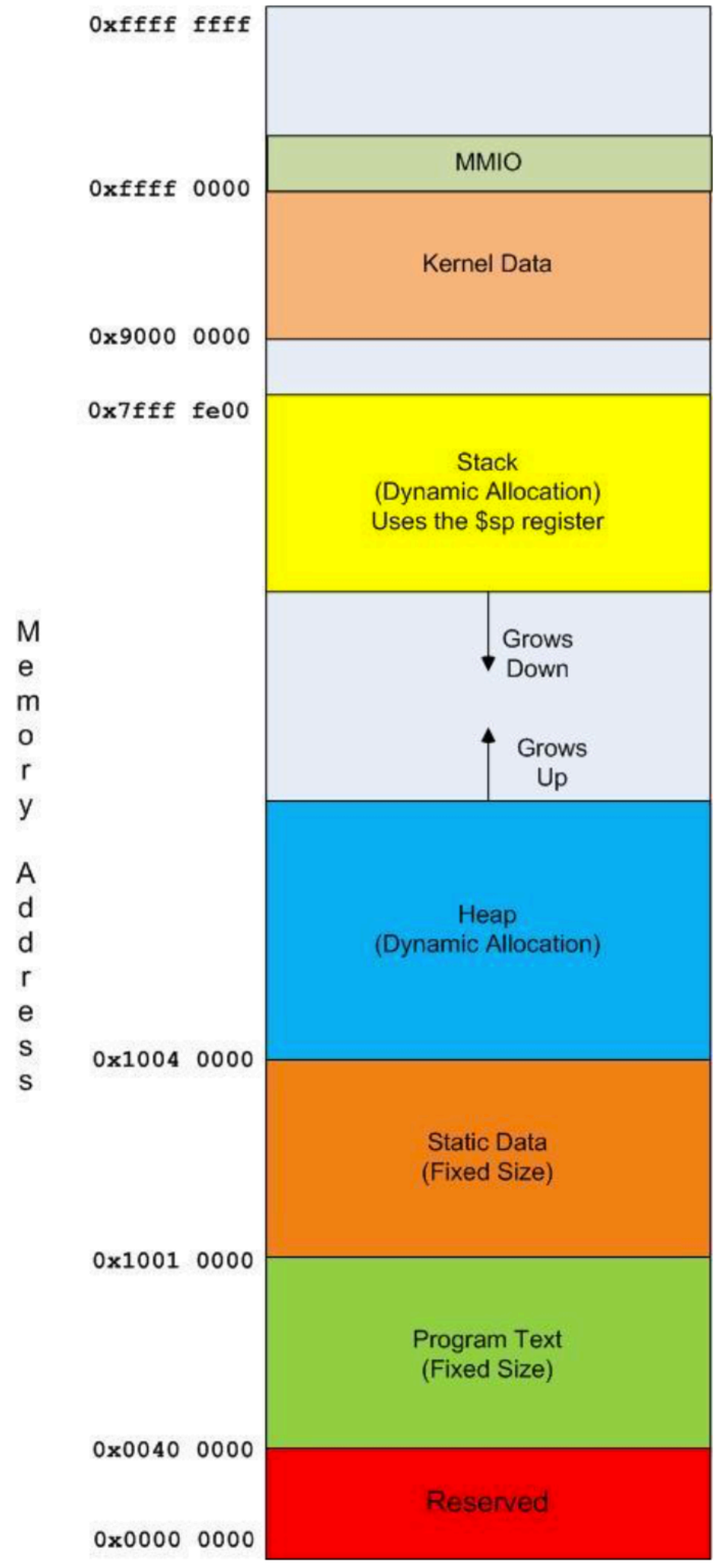
Types of memory

- Static data - (Addresses 0x1001 0000 - 0x1004 0000)
- This is data which will come from the *data segment* of the program.
- The size of the elements in this section are assigned when the program is created (assembled and linked), and cannot change during the execution of the program.



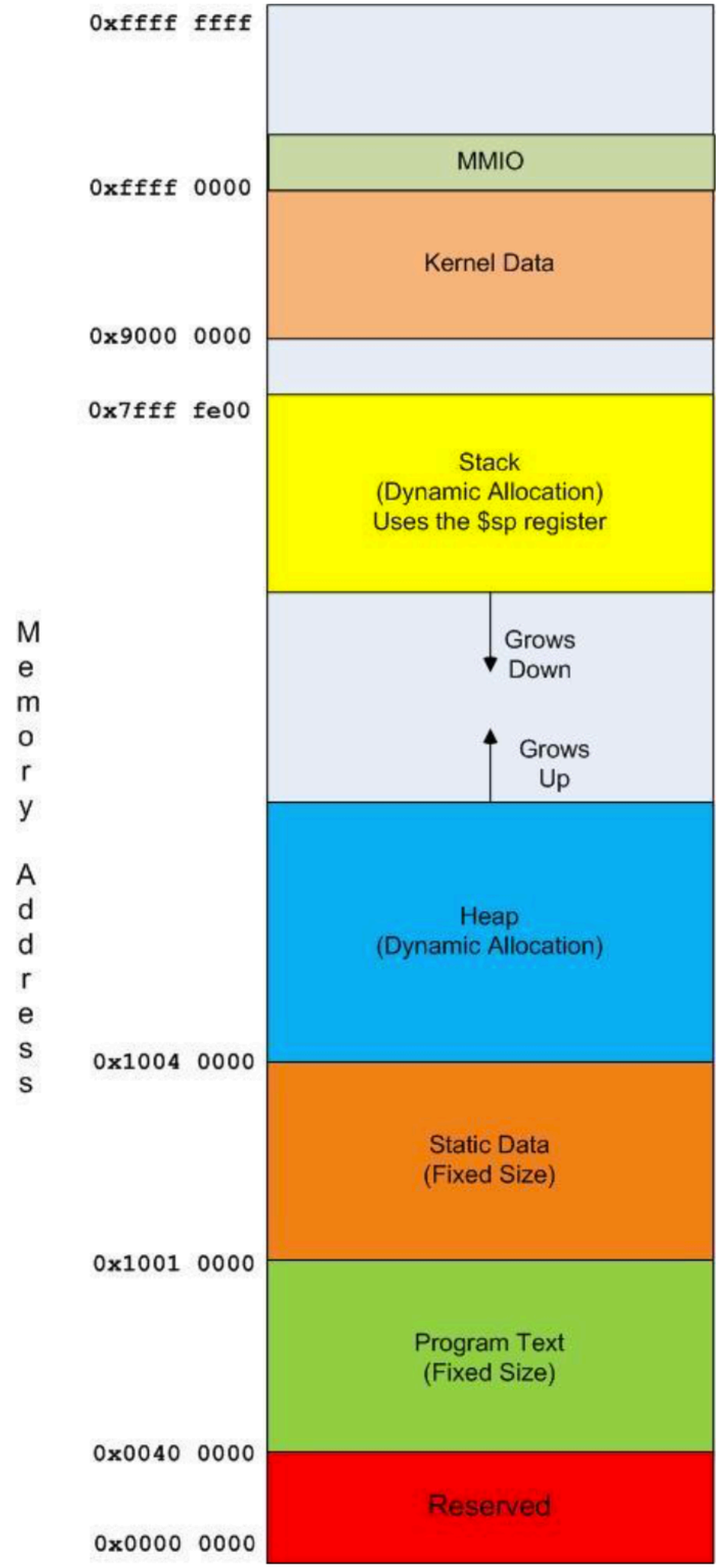
Types of memory

- Heap - (Addresses 0x1004 0000 - until stack data is reached, grows upward)
- Heap is dynamic data which is allocated on an as-needed basis at run time.
- How this memory is allocated and reclaimed is language specific.
- Data in heap is always globally available.



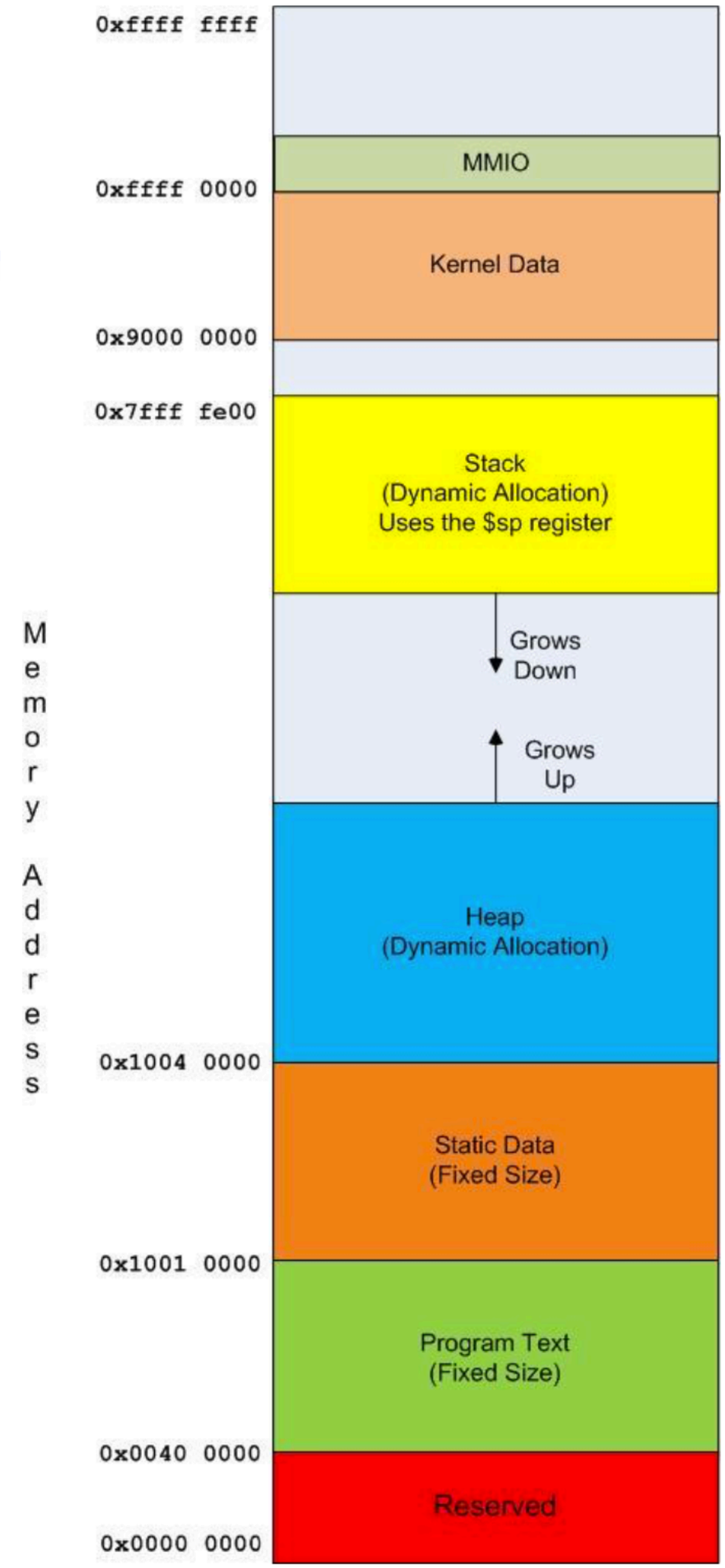
Types of memory

- Stack – (Addresses 0x7fff fe00 - until heap data is reached, grows downward)
- The program stack is dynamic data allocated for subprograms via *push* and *pop* operations.
- All method local variables are stored here. Because of the nature of the push and pop operations, the size of the stack record to create must be known when the program is assembled.



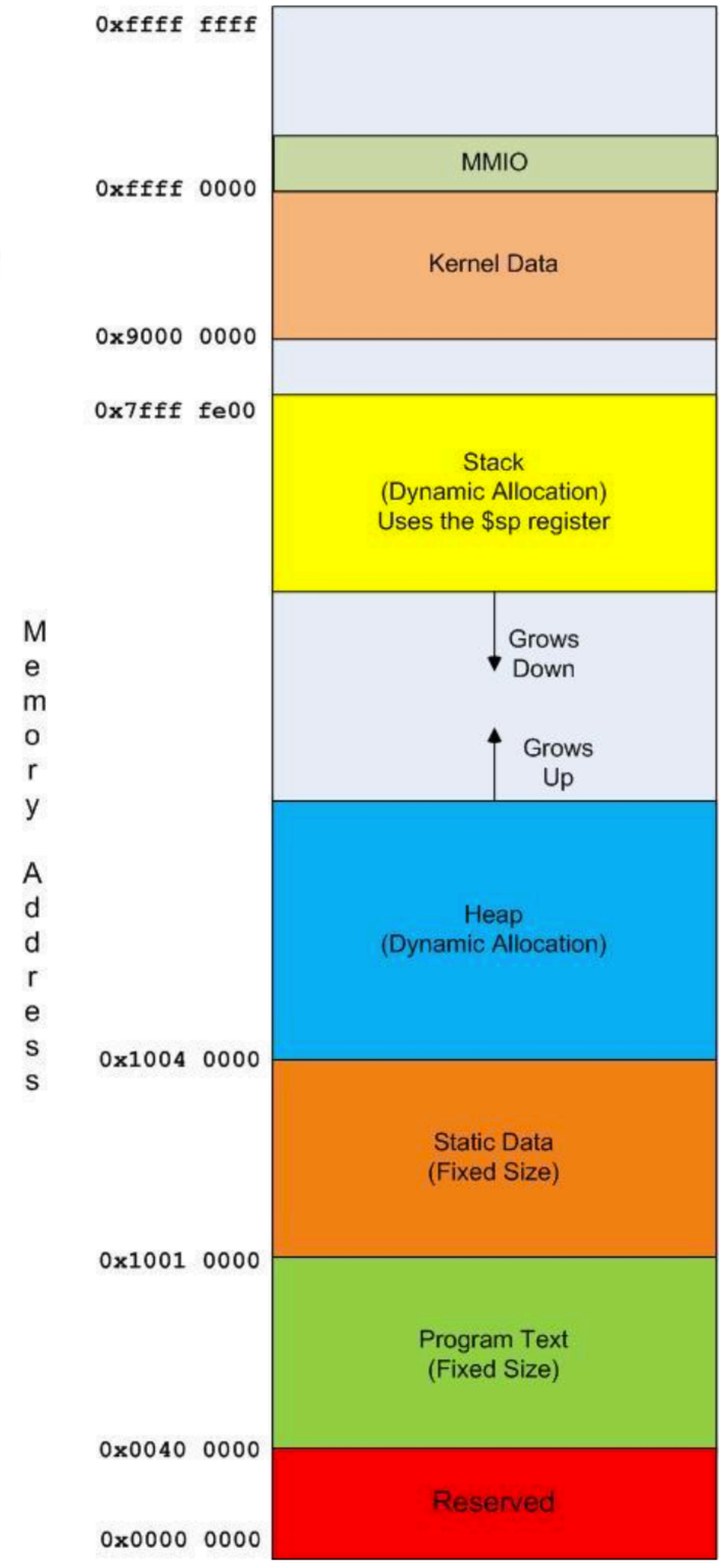
Types of memory

- Kernel - (Addresses 0x9000 0000 - 0xffff 0000)
- Used by the operating system, and so is not accessible to the user.



Types of memory

- MMIO - (Addresses 0xffff 0000 - 0xffff 0010) - Memory Mapped I/O
- Used for any type of external data not in memory, such as monitors, disk drives, consoles, etc.



Program 2.1 Hello World

```
# Program File: Program 2-1.asm
# Author: NTTNga
# Purpose: First program, Hello World
.data                          # Define the program data.
greeting: .asciiz "Hello World" #The string to print.

.text                          # Define the program instructions.
main:                          # Label to define the main program.
        li $v0,4               # Load 4 into $v0 to indicate a print string.
        la $a0,greeting        # Load the address of the greeting into $a0.
        syscall                # Print greeting. The print is indicated by
                               # $v0 having a value of 4, and the string to
                               # print is stored at the address in $a0.
        li $v0,10              # Load a 10 (halt) into $v0.
        syscall                # The program ends.
```

- MIPS assembler code can be indented, and left white space on a line is ignored.

Program 2.1 Hello World

```
# Program File: Program 2-1.asm
# Author: NTTNga
# Purpose: First program, Hello World
.data                           # Define the program data.
greeting: .asciiz "Hello World" #The string to print.

.text                           # Define the program instructions.
main:                           # Label to define the main program.
        li $v0,4                # Load 4 into $v0 to indicate a print string.
        la $a0,greeting         # Load the address of the greeting into $a0.
        syscall                 # Print greeting. The print is indicated by
                                # $v0 having a value of 4, and the string to
                                # print is stored at the address in $a0.
        li $v0,10               # Load a 10 (halt) into $v0.
        syscall                 # The program ends.
```

- All instructions must be on a single line.

Program 2.1 Hello World

```
# Program File: Program 2-1.asm
# Author: NTTNga
# Purpose: First program, Hello World
.data                                   # Define the program data.
greeting: .asciiz "Hello World"         #The string to print.

.text                                   # Define the program instructions.
main:                                   # Label to define the main program.
        li $v0,4                        # Load 4 into $v0 to indicate a print string.
        la $a0,greeting                 # Load the address of the greeting into $a0.
        syscall                         # Print greeting. The print is indicated by
                                        # $v0 having a value of 4, and the string to
                                        # print is stored at the address in $a0.
        li $v0,10                       # Load a 10 (halt) into $v0.
        syscall                         # The program ends.
```

- The # is means any text from the # to the end of a line is a comment and to be ignored.

Program 2.1 Hello World

```
# Program File: Program 2-1.asm
# Author: NTTNga
# Purpose: First program, Hello World
.data                                   # Define the program data.
greeting: .asciiz "Hello World"         #The string to print.

.text                                   # Define the program instructions.
main:                                   # Label to define the main program.
        li $v0,4                        # Load 4 into $v0 to indicate a print string.
        la $a0,greeting                 # Load the address of the greeting into $a0.
        syscall                         # Print greeting. The print is indicated by
                                        # $v0 having a value of 4, and the string to
                                        # print is stored at the address in $a0.
        li $v0,10                       # Load a 10 (halt) into $v0.
        syscall                         # The program ends.
```

- Strings are denoted by “”’s marks around the string.

Program 2.1 Hello World

```
# Program File: Program 2-1.asm
# Author: NTTNga
# Purpose: First program, Hello World
.data                                   # Define the program data.
greeting: .asciiz "Hello World" #The string to print.

.text                                   # Define the program instructions.
main:                                   # Label to define the main program.
        li $v0,4                        # Load 4 into $v0 to indicate a print string.
        la $a0,greeting                 # Load the address of the greeting into $a0.
        syscall                         # Print greeting. The print is indicated by
                                        # $v0 having a value of 4, and the string to
                                        # print is stored at the address in $a0.
        li $v0,10                       # Load a 10 (halt) into $v0.
        syscall                         # The program ends.
```

- Note the comments at the start of the file. These will be called a ***file preamble*** in this text.

- At a minimum all program should contain at least these comments.

Program 2.1 Hello World

```
# Program File: Program 2-1.asm
# Author: NTTNga
# Purpose: First program, Hello World
.data                           # Define the program data.
greeting: .asciiz "Hello World" #The string to print.

.text                           # Define the program instructions.
main:                           # Label to define the main program.
        li $v0,4                # Load 4 into $v0 to indicate a print string.
        la $a0,greeting         # Load the address of the greeting into $a0.
        syscall                 # Print greeting. The print is indicated by
                                # $v0 having a value of 4, and the string to
                                # print is stored at the address in $a0.
        li $v0,10               # Load a 10 (halt) into $v0.
        syscall                 # The program ends.
```

- Assembly language programs are not ***compiled***, they are ***assembled***.

Program 2.1 Hello World

```
# Program File: Program 2-1.asm
# Author: NTTNga
# Purpose: First program, Hello World
.data                           # Define the program data.
greeting: .asciiz "Hello World" #The string to print.

.text                           # Define the program instructions.
main:                           # Label to define the main program.
        li $v0,4                # Load 4 into $v0 to indicate a print string.
        la $a0,greeting         # Load the address of the greeting into $a0.
        syscall                 # Print greeting. The print is indicated by
                                # $v0 having a value of 4, and the string to
                                # print is stored at the address in $a0.
        li $v0,10               # Load a 10 (halt) into $v0.
        syscall                 # The program ends.
```

- A "." before a text string means the token (string) that follows it is an ***assembler directive***.

Program 2.1 Hello World

```
# Program File: Program 2-1.asm
# Author: NTTNga
# Purpose: First program, Hello World
.data                           # Define the program data.
greeting: .asciiz "Hello World" #The string to print.

.text                           # Define the program instructions.
main:                           # Label to define the main program.
        li $v0,4                # Load 4 into $v0 to indicate a print string.
        la $a0,greeting         # Load the address of the greeting into $a0.
        syscall                 # Print greeting. The print is indicated by
                                # $v0 having a value of 4, and the string to
                                # print is stored at the address in $a0.
        li $v0,10               # Load a 10 (halt) into $v0.
        syscall                 # The program ends.
```

- The **.text** directive means the instructions that follow are part of a program text (i.e. the program), and to be assembled into a program and stored in the text region of memory.

Program 2.1 Hello World

```
# Program File: Program 2-1.asm
# Author: NTTNga
# Purpose: First program, Hello World
.data                           # Define the program data.
greeting: .asciiz "Hello World" #The string to print.

.text                           # Define the program instructions.
main:                           # Label to define the main program.
        li $v0,4                # Load 4 into $v0 to indicate a print string.
        la $a0,greeting         # Load the address of the greeting into $a0.
        syscall                 # Print greeting. The print is indicated by
                                # $v0 having a value of 4, and the string to
                                # print is stored at the address in $a0.
        li $v0,10               # Load a 10 (halt) into $v0.
        syscall                 # The program ends.
```

- The **.data** directive means that what follows it is program data, and to be stored in the static data region of memory.

Program 2.1 Hello World

```
# Program File: Program 2-1.asm
# Author: NTTNga
# Purpose: First program, Hello World
.data                           # Define the program data.
greeting: .asciiz "Hello World" #The string to print.

.text                           # Define the program instructions.
main:                           # Label to define the main program.
        li $v0,4                # Load 4 into $v0 to indicate a print string.
        la $a0,greeting         # Load the address of the greeting into $a0.
        syscall                 # Print greeting. The print is indicated by
                                # $v0 having a value of 4, and the string to
                                # print is stored at the address in $a0.
        li $v0,10               # Load a 10 (halt) into $v0.
        syscall                 # The program ends.
```

- The **.asciiz** directive tells the assembler to interpret the data which follows it as an ASCII string.

Program 2.1 Hello World

```
# Program File: Program 2-1.asm
# Author: NTTNga
# Purpose: First program, Hello World
.data                           # Define the program data.
greeting: .asciiz "Hello World" #The string to print.

.text                           # Define the program instructions.
main:                           # Label to define the main program.
        li $v0,4                # Load 4 into $v0 to indicate a print string.
        la $a0,greeting         # Load the address of the greeting into $a0.
        syscall                 # Print greeting. The print is indicated by
                                # $v0 having a value of 4, and the string to
                                # print is stored at the address in $a0.
        li $v0,10               # Load a 10 (halt) into $v0.
        syscall                 # The program ends.
```

- In MIPS assembler any text string followed by a ":" is a label.
- A label is just a marker in the code that can be used in other statements.

Program 2.1 Hello World

```
# Program File: Program 2-1.asm
# Author: NTTNga
# Purpose: First program, Hello World
.data                           # Define the program data.
greeting: .asciiz "Hello World" #The string to print.

.text                           # Define the program instructions.
main:                           # Label to define the main program.
        li $v0,4                # Load 4 into $v0 to indicate a print string.
        la $a0,greeting         # Load the address of the greeting into $a0.
        syscall                 # Print greeting. The print is indicated by
                                # $v0 having a value of 4, and the string to
                                # print is stored at the address in $a0.
        li $v0,10               # Load a 10 (halt) into $v0.
        syscall                 # The program ends.
```

- The label **main**: does not need to be included as MARS assumes the program begins at the first line in the assembled program. But it is nice to label the starting point, and generally most runtimes will look for a global symbol name ***main*** as the place to begin execution.

Program 2.1 Hello World

```
# Program File: Program 2-1.asm
# Author: NTTNga
# Purpose: First program, Hello World
.data                           # Define the program data.
greeting: .asciiz "Hello World" #The string to print.

.text                           # Define the program instructions.
main:                           # Label to define the main program.
        li $v0,4                # Load 4 into $v0 to indicate a print string.
        la $a0,greeting         # Load the address of the greeting into $a0.
        syscall                 # Print greeting. The print is indicated by
                                # $v0 having a value of 4, and the string to
                                # print is stored at the address in $a0.
        li $v0,10               # Load a 10 (halt) into $v0.
        syscall                 # The program ends.
```

- Any time a constant is included in an instruction, it is called an ***immediate*** value.
- The constant must be in the instruction itself.

Program 2.1 Hello World

```
# Program File: Program 2-1.asm
# Author: NTTNga
# Purpose: First program, Hello World
.data                           # Define the program data.
greeting: .asciiz "Hello World" #The string to print.

.text                           # Define the program instructions.
main:                           # Label to define the main program.
        li $v0,4                # Load 4 into $v0 to indicate a print string.
        la $a0,greeting         # Load the address of the greeting into $a0.
        syscall                 # Print greeting. The print is indicated by
                                # $v0 having a value of 4, and the string to
                                # print is stored at the address in $a0.
        li $v0,10               # Load a 10 (halt) into $v0.
        syscall                 # The program ends.
```

- Only **instructions** and **labels** can be defined in a text segment
- Only **data** and **labels** can be defined in a data segment.

Program 2.1 Hello World

```
# Program File: Program 2-1.asm
# Author: NTTNga
# Purpose: First program, Hello World
.data                           # Define the program data.
greeting: .asciiz "Hello World" #The string to print.

.text                           # Define the program instructions.
main:                           # Label to define the main program.
        li $v0,4                # Load 4 into $v0 to indicate a print string.
        la $a0,greeting         # Load the address of the greeting into $a0.
        syscall                 # Print greeting. The print is indicated by
                                # $v0 having a value of 4, and the string to
                                # print is stored at the address in $a0.
        li $v0,10               # Load a 10 (halt) into $v0.
        syscall                 # The program ends.
```

- Operators are text strings like **li**, **la**, and **syscall**.

Program 2.1 Hello World

```
# Program File: Program 2-1.asm
# Author: NTTNga
# Purpose: First program, Hello World
.data                          # Define the program data.
greeting: .asciiz "Hello World" #The string to print.

.text                          # Define the program instructions.
main:                          # Label to define the main program.
        li $v0,4               # Load 4 into $v0 to indicate a print string.
        la $a0,greeting        # Load the address of the greeting into $a0.
        syscall                # Print greeting. The print is indicated by
                               # $v0 having a value of 4, and the string to
                               # print is stored at the address in $a0.
        li $v0,10              # Load a 10 (halt) into $v0.
        syscall                # The program ends.
```

- Instructions are operators and their arguments.
- So **li** is an operator; **li $v0, 4** is an instruction.

Program 2.1 Hello World

```
# Program File: Program 2-1.asm
# Author: NTTNga
# Purpose: First program, Hello World
.data                           # Define the program data.
greeting: .asciiz "Hello World" #The string to print.

.text                           # Define the program instructions.
main:                           # Label to define the main program.
        li $v0,4                # Load 4 into $v0 to indicate a print string.
        la $a0,greeting         # Load the address of the greeting into $a0.
        syscall                 # Print greeting. The print is indicated by
                                # $v0 having a value of 4, and the string to
                                # print is stored at the address in $a0.
        li $v0,10               # Load a 10 (halt) into $v0.
        syscall                 # The program ends.
```

- The syscall operator is used to call system services.

Look up in HELP

MARS 4.5 Help

MIPS | MARS | License | Bugs/Comments | Acknowledgements | Instruction Set Song

Operand Key for Example Instructions

`label, target`	any textual label
`$t1, $t2, $t3`	any integer register
`$f2, $f4, $f6`	*even-numbered* floating point register
`$f0, $f1, $f3`	*any* floating point register

Basic Instructions | Extended (pseudo) Instructions | Directives | Syscalls | Exceptions | Macros

```
li  $v0, 1            # service 1 is print integer
add $a0, $t0, $zero   # load desired value into argument register $a0, using pseudo-op
syscall
```

Table of Available Services

Service	Code in $v0	Arguments	Result
print integer	1	$a0 = integer to print	
print float	2	$f12 = float to print	
print double	3	$f12 = double to print	
print string	4	$a0 = address of null-terminated string to print	
read integer	5		$v0 contains integer read
read float	6		$f0 contains float read
read double	7		$f0 contains double read
read string	8	$a0 = address of input buffer $a1 = maximum number of characters to read	*See note below table*

Close

Program 2.2 - Prompt and read an integer from a user

```
# Program File: Program 2-2.asm
# Author: NTTNga
# Program to read an integer number from a user, and
# print that number back to the console.

.data
prompt: .asciiz "Hay nhap vao mot so nguyen: "
output: .asciiz "\nBan da nhap vao so: "

.text
main:
        # Prompt for the integer to enter
        li $v0,4
        la $a0,prompt
        syscall

        # Read the integer and save it in $s0
        li $v0,5
        syscall
        move $s0,$v0

        # Output the text
        li $v0,4
        la $a0,output
        syscall

        # Output the number
        li $v0,1
        move $a0,$s0
        syscall

        # Exit the program
        li $v0,10
        syscall
```

- Blocks of code are commented, not each individual statement.
- Each block should be commented as to what it does, and/or how the code works.

Program 2.2 - Prompt and read an integer from a user

```
# Program File: Program 2-2.asm
# Author: NTTNga
# Program to read an integer number from a user, and
# print that number back to the console.

.data
prompt: .asciiz "Hay nhap vao mot so nguyen: "
output: .asciiz "\nBan da nhap vao so: "

.text
main:
        # Prompt for the integer to enter
        li $v0,4
        la $a0,prompt
        syscall

        # Read the integer and save it in $s0
        li $v0,5
        syscall
        move $s0,$v0

        # Output the text
        li $v0,4
        la $a0,output
        syscall

        # Output the number
        li $v0,1
        move $a0,$s0
        syscall

        # Exit the program
        li $v0,10
        syscall
```

- The **move** operator is introduced.
- The **move** operator moves the text from one register to another.

Program 2.2 - Prompt and read an integer from a user

```
# Program File: Program 2-2.asm
# Author: NTTNga
# Program to read an integer number from a user, and
# print that number back to the console.

.data
prompt: .asciiz "Hay nhap vao mot so nguyen: "
output: .asciiz "\nBan da nhap vao so: "

.text
main:
        # Prompt for the integer to enter
        li $v0,4
        la $a0,prompt
        syscall

        # Read the integer and save it in $s0
        li $v0,5
        syscall
        move $s0,$v0

        # Output the text
        li $v0,4
        la $a0,output
        syscall

        # Output the number
        li $v0,1
        move $a0,$s0
        syscall

        # Exit the program
        li $v0,10
        syscall
```

- **Service 5** synchronously waits for the user to enter an integer on the console
- When the integer is typed, it is returned to the return register **$v0**.
- This service checks to see that the value entered is an integer value.
- Raises an exception if it is not.

Program 2.2 - Prompt and read an integer from a user

```
# Program File: Program 2-2.asm
# Author: NTTNga
# Program to read an integer number from a user, and
# print that number back to the console.

.data
prompt: .asciiz "Hay nhap vao mot so nguyen: "
output: .asciiz "\nBan da nhap vao so: "

.text
main:
        # Prompt for the integer to enter
        li $v0,4
        la $a0,prompt
        syscall

        # Read the integer and save it in $s0
        li $v0,5
        syscall
        move $s0,$v0

        # Output the text
        li $v0,4
        la $a0,output
        syscall

        # Output the number
        li $v0,1
        move $a0,$s0
        syscall

        # Exit the program
        li $v0,10
        syscall
```

- Service 1 prints out the integer value in register **$a0**.
- Note:
  - With **service 4**, string that is at the address in **$a0** (or referenced by **$a0**) is printed.
  - With the **service 1** the value in register **$a0** is printed.

Program 2.2 - Prompt and read an integer from a user

```
# Program File: Program 2-2.asm
# Author: NTTNga
# Program to read an integer number from a user, and
# print that number back to the console.

.data
prompt: .asciiz "Hay nhap vao mot so nguyen: "
output: .asciiz "\nBan da nhap vao so: "

.text
main:
        # Prompt for the integer to enter
        li $v0,4
        la $a0,prompt
        syscall

        # Read the integer and save it in $s0
        li $v0,5
        syscall
        move $s0,$v0

        # Output the text
        li $v0,4
        la $a0,output
        syscall

        # Output the number
        li $v0,1
        move $a0,$s0
        syscall

        # Exit the program
        li $v0,10
        syscall
```

- An escape character "**\n**" is used in the string named **output**.
- This escape character is called the new line character, causes the **output** from the program to start on the next line.

Program 2.3 - Prompt and read a string from a user

```
# Program File: Program 2-3.asm
# Author: NTTNga
# Program to read a string from a user, and
# print that string back to the console.

.data
input: .space 81
inputSize: .word 80
prompt: .asciiz "Hay nhap vao mot chuoi: "
output: .asciiz "\nBan da nhap vao chuoi: "

.text
main:
   # Prompt for the string to enter
   li $v0,4
   la $a0,prompt
   syscall

   # Read the string.
   li $v0,8
   la $a0,input
   lw $a1,inputSize
   syscall

   # Output the text
   li $v0,4
   la $a0,output
   syscall

   # Output the number
   li $v0,4
   la $a0,input
   syscall

   # Exit the program
   li $v0,10
   syscall
```

- The **.space** directive allocates **n** bytes of memory in the data region of the program, where n=81 in this program.
- Since the size of a character is **1 byte**, this is equivalent to saving 80 characters for data.
- Why 81 bytes of memory is declared?

Program 2.3 - Prompt and read a string from a user

```
# Program File: Program 2-3.asm
# Author: NTTNga
# Program to read a string from a user, and
# print that string back to the console.

.data
input: .space 81
inputSize: .word 80
prompt: .asciiz "Hay nhap vao mot chuoi: "
output: .asciiz "\nBan da nhap vao chuoi: "

.text
main:
   # Prompt for the string to enter
   li $v0,4
   la $a0,prompt
   syscall

   # Read the string.
   li $v0,8
   la $a0,input
   lw $a1,inputSize
   syscall

   # Output the text
   li $v0,4
   la $a0,output
   syscall

   # Output the number
   li $v0,4
   la $a0,input
   syscall

   # Exit the program
   li $v0,10
   syscall
```

- The **.word** directive allocates 4 bytes of space in the data region.
- The **.word** directive can then be given an integer value, and it will initialize the allocated space to that integer value.
- What is stored in this memory can by any type of data.

Program 2.3 - Prompt and read a string from a user

```
# Program File: Program 2-3.asm
# Author: NTTNga
# Program to read a string from a user, and
# print that string back to the console.

.data
input: .space 81
inputSize: .word 80
prompt: .asciiz "Hay nhap vao mot chuoi: "
output: .asciiz "\nBan da nhap vao chuoi: "

.text
main:
   # Prompt for the string to enter
   li $v0,4
   la $a0,prompt
   syscall

   # Read the string.
   li $v0,8
   la $a0,input
   lw $a1,inputSize
   syscall

   # Output the text
   li $v0,4
   la $a0,output
   syscall

   # Output the number
   li $v0,4
   la $a0,input
   syscall

   # Exit the program
   li $v0,10
   syscall
```

- The **la** operator loads the address of the label into a register.
- Called a reference to the data
- Be shown in the text as:

  $a0 <= label

  which means the value of the label (the memory address) is loaded into a register.

Program 2.3 - Prompt and read a string from a user

```
# Program File: Program 2-3.asm
# Author: NTTNga
# Program to read a string from a user, and
# print that string back to the console.

.data
input: .space 81
inputSize: .word 80
prompt: .asciiz "Hay nhap vao mot chuoi: "
output: .asciiz "\nBan da nhap vao chuoi: "

.text
main:
   # Prompt for the string to enter
   li $v0,4
   la $a0,prompt
   syscall

   # Read the string.
   li $v0,8
   la $a0,input
   lw $a1,inputSize
   syscall

   # Output the text
   li $v0,4
   la $a0,output
   syscall

   # Output the number
   li $v0,4
   la $a0,input
   syscall

   # Exit the program
   li $v0,10
   syscall
```

- The **lw** operator loads the value contained at the label into the register.
- Loading of values into a register will be shown as:

$a1 <= M[label]

which means the value at the label is loaded into register **$a1**.

Program 2.3 - Prompt and read a string from a user

Address	Value (+0)	Value (+4)
0x10010000	l l e H	\0 \0 \n o
0x10010020	\0 \0 \0 \0	\0 \0 \0 \0
0x10010040	\0 \0 \0 \0	\0 \0 \0 \0
0x10010060	o a v	t o m
0x10010080	h c o	: i o u
0x100100a0	\0 \0 \0 \0	\0 \0 \0 \0
0x100100c0	\0 \0 \0 \0	\0 \0 \0 \0

Address	Value (+0)	Value (+4)
0x10010000	0x6c6c6548	0x00000a6f
0x10010020	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000
0x10010060	0x6f617620	0x746f6d20
0x10010080	0x6863206f	0x3a696f75
0x100100a0	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000

- A string is a sequence of ASCII characters which are terminated with a **null** value.
- When handling strings, an extra byte must always be added to include the null terminator.

Program 2.3 - Prompt and read a string from a user

```
# Program File: Program 2-3.asm
# Author: NTTNga
# Program to read a string from a user, and
# print that string back to the console.

.data
input: .space 81
inputSize: .word 80
prompt: .asciiz "Hay nhap vao mot chuoi: "
output: .asciiz "\nBan da nhap vao chuoi: "

.text
main:
    # Prompt for the string to enter
    li $v0,4
    la $a0,prompt
    syscall

    # Read the string.
    li $v0,8
    la $a0,input
    lw $a1,inputSize
    syscall

    # Output the text
    li $v0,4
    la $a0,output
    syscall

    # Output the number
    li $v0,4
    la $a0,input
    syscall

    # Exit the program
    li $v0,10
    syscall
```

- Syscall **service 8** is used to read a string from the console.
- There are two parameters passed to the service:
  - A reference to the memory to use to store the string (stored in **$a0**)
  - The maximum size of the string to read (stored in **$a1**)

Program 2.3 - Prompt and read a string from a user

Address	Value (+0)	Value (+4)	Value (+8)
0x10010000	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x10010020	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x10010040	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x10010060	o a v	t o m	u h c

Address	Value (+0)	Value (+4)	Value (+8)
0x10010000	l l e H	\0 \0 \n o	\0 \0 \0 \0
0x10010020	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x10010040	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x10010060	o a v	t o m	u h c
0x10010080	h c o	: i o u	\0 \0 \0

Assigment 1

Gõ chương trình sau vào công cụ MARS.

```
#Laboratory Exercise 2, Assignment 2
.text
      lui    $s0,0x2110          #put upper half of pattern in $s0
      ori    $s0,$s0,0x003d      #put lower half of pattern in $s0
```

Sau đó:

- Sử dụng công cụ gỡ rối, Debug, chạy từng lệnh và dừng lại,
- Ở mỗi lệnh, quan sát cửa sổ Register và chú ý
    - Sự thay đổi giá trị của thanh ghi $s0
    - Sự thay đổi giá trị của thanh ghi $pc
- Ở cửa sổ Data Segment, hãy click vào hộp combo để chuyển tới quan sát các byte trong vùng lệnh .text.
    - Kiểm tra xem các byte đầu tiên ở vùng lệnh trùng với cột nào trong cửa sổ Text Segment.

Assigment 2

Gõ chương trình sau vào công cụ MARS.

```
#Laboratory Exercise 2, Assignment 3
.text
      li    $s0,0x2110003d #pseudo instruction=2 basic instructions
      li    $s1,0x2      #but if the immediate value is small, one ins
```

Sau đó:

- Biên dịch và quan sát các lệnh mã máy trong cửa sổ Text Segment. Giải thích điều bất thường?

Assigment 3

- Write a program which prompts the user to enter their favorite type of pie.
- The program should then print out "So you like _____ pie", where the blank line is replaced by the pie type entered.
- What annoying feature of syscall service 4 makes it impossible at this point to make the output appear on a single line?

End of week 2