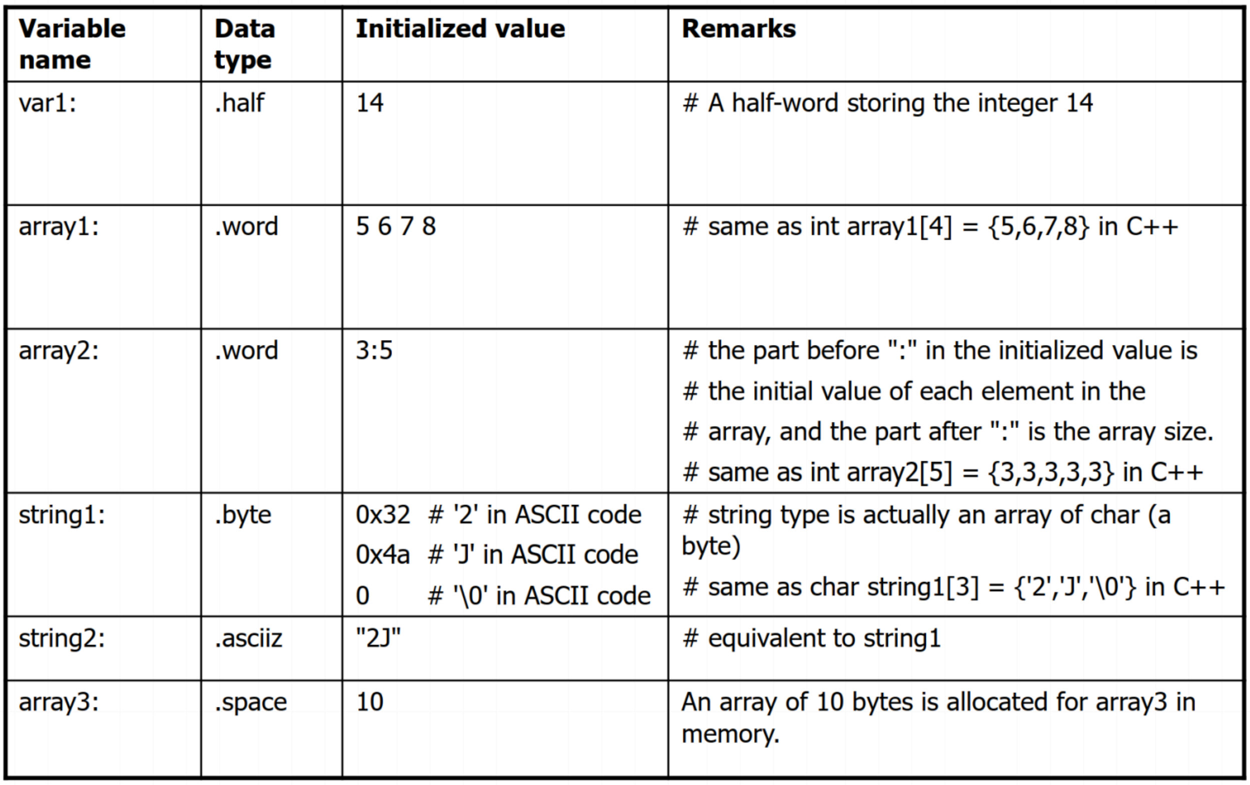
COMP2611 Computer Organization

MIPS Review Notes

1. Data Types

```
# Data segment
.data
h: .word 1 2 3 4  # h is an array of size 4, each element is a word (32 bit)
s: .word 3:5  # s is an array of size 5, each element equals to 3. s[]=
{3,3,3,3,3}
fourbytes: .ascii "12AB"  # .ascii: string without null terminator
message: .asciiz "\nHello World!\n"  # .asciiz: String with terminator


# Program begins
.text
.globl __start
__start:
```



Variable name	Data type	Initialized value	Remarks
var1:	.half	14	# A half-word storing the integer 14
array1:	.word	5 6 7 8	# same as int array1[4] = {5,6,7,8} in C++
array2:	.word	3:5	# the part before ":" in the initialized value is # the initial value of each element in the # array, and the part after ":" is the array size. # same as int array2[5] = {3,3,3,3,3} in C++
string1:	.byte	0x32 # '2' in ASCII code 0x4a # 'J' in ASCII code 0 # '\0' in ASCII code	# string type is actually an array of char (a byte) # same as char string1[3] = {'2','J','\0'} in C++
string2:	.asciiz	"2J"	# equivalent to string1
array3:	.space	10	An array of 10 bytes is allocated for array3 in memory.

2. Arithmetic Operations

```
add   $a, $b, $c    # $a = $b + $c
sub   $a, $b, $c    # $a = $b - $c
addi  $a, $b, 2     # $a = $b + 2
addi  $a, $b, -2    # $a = $b - 2

sll $a$, $b$, 2   # a = b << 2 = b * 4
srl $a$, $b$, 2   # a = b >> 2 = b // 4
sra $a$, $b$, 2   # shift right arithmetic, sign extension
```

3. Logic Operations

```
addi $s0, $0, 0xffff0000
addi $s1, $0, 0xaaaa1111
not $t0, $s0
and $t1, $s0, $s1
or $t2, $s0, $s1
nor $t3, $s0, $s1
xor $t4, $s0, $s1

# Expected Result:
# $t0 = 0x0000ffff
# $t1 = 0xaaaa0000
# $t2 = 0xffff1111
# $t3 = 0x0000eeee
# $t4 = 0x55551111
```

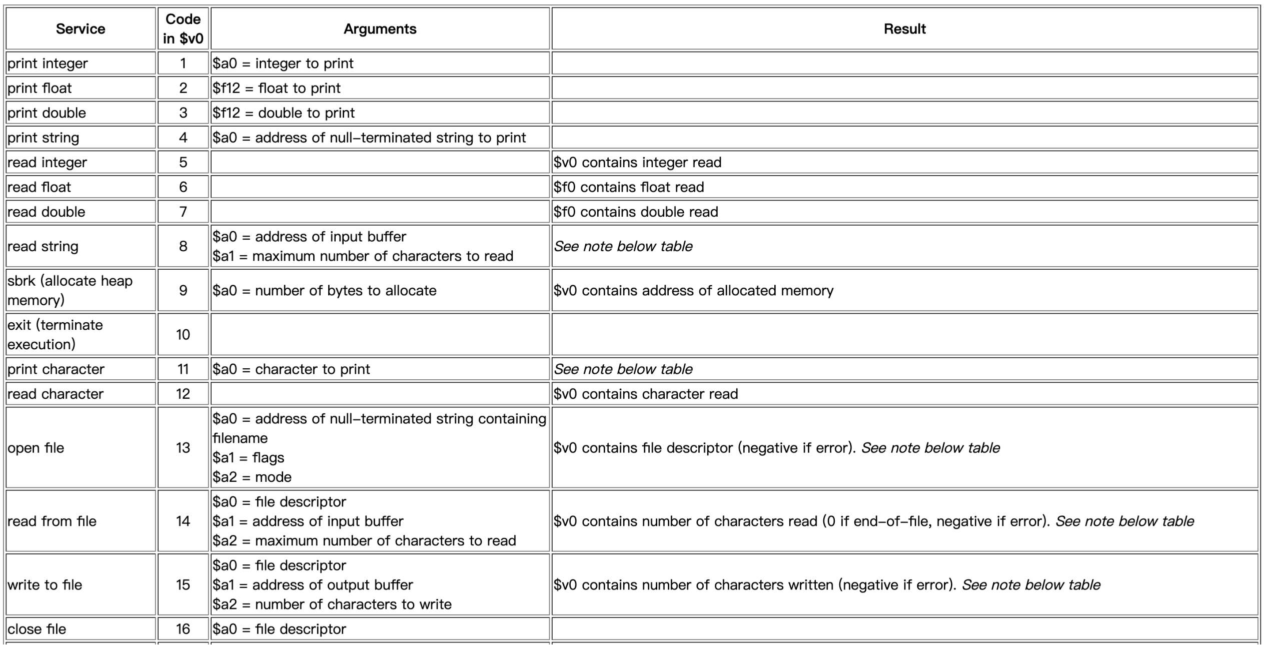
4. Data Transfer

```
.data
fourbytes: .ascii "12AB"
fourwords: .word 1 -1 1024 -65536

.text
.globl _main
_main:
la $s0, fourbytes   # load address
lb $t0, 0($s0)      # load_byte, $t0 = '1' = 49
lb $t1, 2($s0)      # $t1 = 'A' = 65
lb $t2, 1($s0)      # $t2 = '2' = 50
lb $t3, 4($s0)      # out of range, $t3 = '1' = 1

la $s1, fourwords
lw $t0, 0($s1)      # $t0 = 1 = 0x00000001
lw $t1, 4($s1)      # $t1 = -1 = 0xffffffff
lw $t2, 8($s1)      # $t0 = 1024 = 0x00000400
lw $t3, 12($s1)     # $t1 = -65536 = 0xffff0000



# Data transfer
# Big-endian: the end of a word matches a bigger address
sw    $t0, 100($s0) # Memory[$s0 + 100] = $t0; store word from reg to mem
sb    $t0, 100($s0) # store rightmost byte in $t0
```

5. Control Flow

```
# if-else statement
# if($s3 == $s4) If...
# else if($s3 == $s1) ElseIf...
# else Else...
# Exit...
  beq $s3, $s4, If
  beq $s3, $s1, ElseIf
  j Else
If: add $s0, $s1, $s2
  j exit
ElseIf: sub $s0, $s1, $s2
  j exit
Else: add $s0, $s1, $s4
  exit:

# Comparison:
# Refer to midterm-quick-reference

# while loop
Loop: bne $t0, $s2, Exit   # go to Exit if $t0 != $s2
  # ...
  addi $s1, $s1, 1    # $s1 = $s1 + 1
  j Loop
Exit:


# Branch comparison with zero
bgez $s, label  # if ($s >= 0)
bgtz $s, label  # if ($s > 0)
blez $s, label  # if ($s <= 0)
bltz $s, label  # if ($s < 0)
```

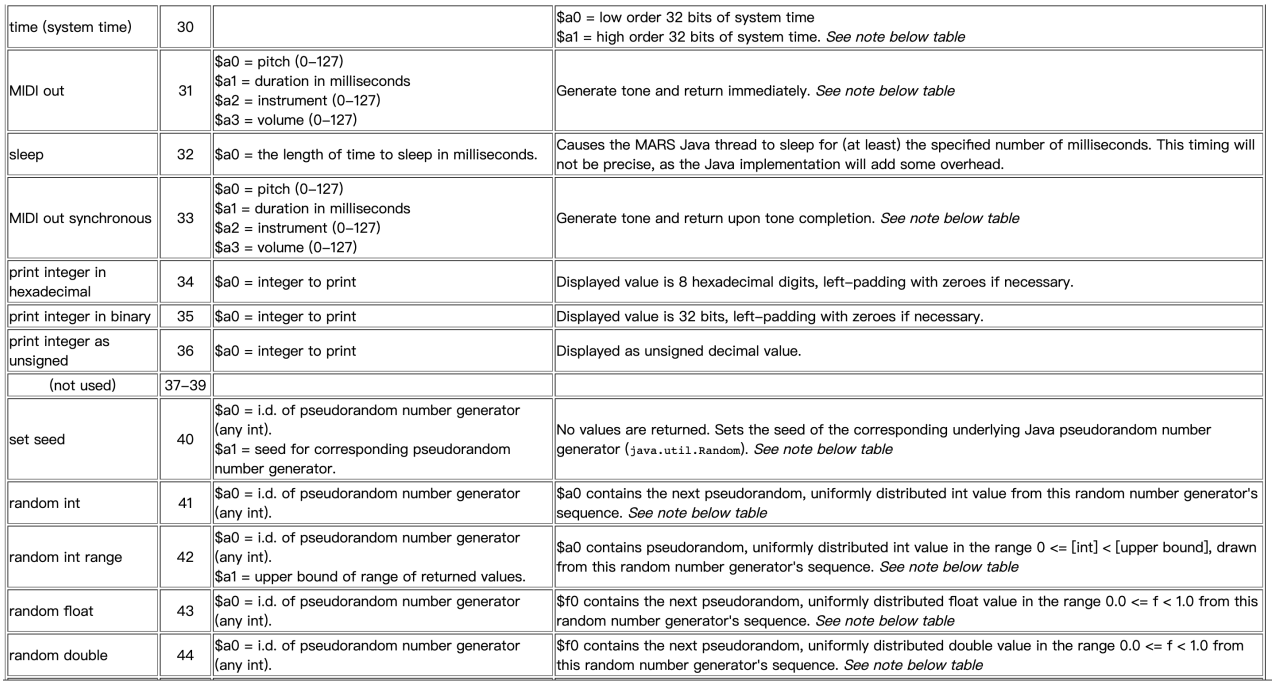
6. System Call



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*
sbrk (allocate heap memory)	9	$a0 = number of bytes to allocate	$v0 contains address of allocated memory
exit (terminate execution)	10		
print character	11	$a0 = character to print	*See note below table*
read character	12		$v0 contains character read
open file	13	$a0 = address of null-terminated string containing filename $a1 = flags $a2 = mode	$v0 contains file descriptor (negative if error). *See note below table*
read from file	14	$a0 = file descriptor $a1 = address of input buffer $a2 = maximum number of characters to read	$v0 contains number of characters read (0 if end-of-file, negative if error). *See note below table*
write to file	15	$a0 = file descriptor $a1 = address of output buffer $a2 = number of characters to write	$v0 contains number of characters written (negative if error). *See note below table*
close file	16	$a0 = file descriptor	

```
# Example of a+b problem:
.data
.text
.globl _main
_main:
li $v0, 5  # syscall code 5: read integer
syscall
add $t0, $zero, $v0

li $v0, 5
syscall
add $t1, $zero, $v0

add $a0, $t0, $t1  # put sum into $a0, which is syscall's parameter

li $v0, 1  # syscall code 1: print integer
syscall

# don't forget to exit
li $v0, 10
syscall
```

Some Syscall related to random: (**30, 40, 42**)



time (system time)	30		$a0 = low order 32 bits of system time $a1 = high order 32 bits of system time. *See note below table*
MIDI out	31	$a0 = pitch (0–127) $a1 = duration in milliseconds $a2 = instrument (0–127) $a3 = volume (0–127)	Generate tone and return immediately. *See note below table*
sleep	32	$a0 = the length of time to sleep in milliseconds.	Causes the MARS Java thread to sleep for (at least) the specified number of milliseconds. This timing will not be precise, as the Java implementation will add some overhead.
MIDI out synchronous	33	$a0 = pitch (0–127) $a1 = duration in milliseconds $a2 = instrument (0–127) $a3 = volume (0–127)	Generate tone and return upon tone completion. *See note below table*
print integer in hexadecimal	34	$a0 = integer to print	Displayed value is 8 hexadecimal digits, left–padding with zeroes if necessary.
print integer in binary	35	$a0 = integer to print	Displayed value is 32 bits, left–padding with zeroes if necessary.
print integer as unsigned	36	$a0 = integer to print	Displayed as unsigned decimal value.
(not used)	37–39		
set seed	40	$a0 = i.d. of pseudorandom number generator (any int). $a1 = seed for corresponding pseudorandom number generator.	No values are returned. Sets the seed of the corresponding underlying Java pseudorandom number generator (`java.util.Random`). *See note below table*
random int	41	$a0 = i.d. of pseudorandom number generator (any int).	$a0 contains the next pseudorandom, uniformly distributed int value from this random number generator's sequence. *See note below table*
random int range	42	$a0 = i.d. of pseudorandom number generator (any int). $a1 = upper bound of range of returned values.	$a0 contains pseudorandom, uniformly distributed int value in the range 0 <= [int] < [upper bound], drawn from this random number generator's sequence. *See note below table*
random float	43	$a0 = i.d. of pseudorandom number generator (any int).	$f0 contains the next pseudorandom, uniformly distributed float value in the range 0.0 <= f < 1.0 from this random number generator's sequence. *See note below table*
random double	44	$a0 = i.d. of pseudorandom number generator (any int).	$f0 contains the next pseudorandom, uniformly distributed double value in the range 0.0 <= f < 1.0 from this random number generator's sequence. *See note below table*

```
# Example: Generate random int with syscall
.data
.text
.globl _main
_main:
li $v0, 30  # get system time
syscall

add $a1, $0, $a0  # put time to random seed
li $a0, 0  # use #0 generator
li $v0, 40
syscall

li $a0, 0
li $a1, 20  # generate [1,20], set upper bound = 20
li $v0, 42
syscall

addi $a0, $a0, 1  # [0,19] -> [1,20]

li $v0, 1  # print integer
syscall
```

7. Procedure

```
To be finished
```