MARS SIMULATOR

- A java based simulator.
- Editor, assembler and debugger.
- Useful as many embedded systems run on MIPS processor.
- All instructions are 32 bits.

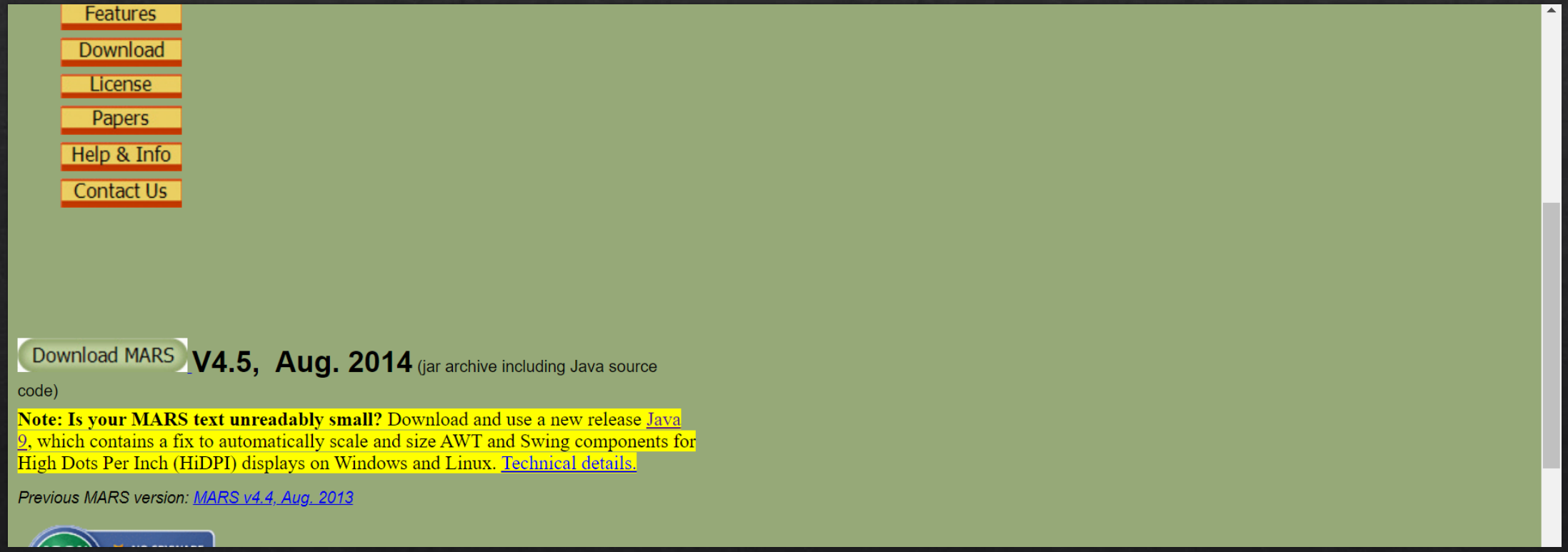
JAVA JDK INSTALLATION

Java SE Development Kit 15.0.2

This software is licensed under the Oracle Technology Network License Agreement for Oracle Java SE

Product / File Description	File Size	Download
Linux ARM 64 RPM Package	141.82 MB	jdk-15.0.2_linux-aarch64_bin.rpm
Linux ARM 64 Compressed Archive	157 MB	jdk-15.0.2_linux-aarch64_bin.tar.gz
Linux x64 Debian Package	154.81 MB	jdk-15.0.2_linux-x64_bin.deb
Linux x64 RPM Package	162.03 MB	jdk-15.0.2_linux-x64_bin.rpm
Linux x64 Compressed Archive	179.35 MB	jdk-15.0.2_linux-x64_bin.tar.gz
macOS Installer	175.93 MB	jdk-15.0.2_osx-x64_bin.dmg
macOS Compressed Archive	176.51 MB	jdk-15.0.2_osx-x64_bin.tar.gz

MARS INSTALLATION



- The simulator is split into 3 segments – Editor, I/O bar and list of registers.
- All the programs will have a .asm extension.
- As these are assembly level language programs, they are assembled not compiled.

REGISTERS IN MARS

Name	Number	Use
$zero	$0	constant 0
$at	$1	assembler temporary
$v0–$v1	$2–$3	values for function returns and expression evaluation
$a0–$a7	$4–$11	function arguments
$t4–$t7	$12–$15	temporaries
$s0–$s7	$16–$23	saved temporaries
$t8–$t9	$24–$25	temporaries
$k0–$k1	$26–$27	reserved for OS kernel
$gp	$28	global pointer
$sp	$29	stack pointer
$s8	$30	frame pointer
$ra	$31	return address

MISP PROGRAM

- MIPS program has 2 sections – data and text.
- Data declaration section - .data Variables created and defined.
- Code section- .text Instructions that need to be executed.
- Start of the code section is label main end is exit syscall. Involves manipulation of registers and performance of arithmetic operations.

MIPS SYSTEM CALLS

Table: System services.

Service	System Call Code	Arguments	Result
print_int	1	`$a0` = integer	
print_float	2	`$f12` = float	
print_double	3	`$f12` = double	
print_string	4	`$a0` = string	
read_int	5		integer (in `$v0`)
read_float	6		float (in `$f0`)
read_double	7		double (in `$f0`)
read_string	8	`$a0` = buffer, `$a1` = length	
sbrk	9	`$a0` = amount	address (in `$v0`)
exit	10		
print_character	11	`$a0` = character	
read_character	12		character (in `$v0`)
open	13	`$a0` = filename,	file descriptor (in `$v0`)
		`$a1` = flags, `$a2` = mode	
read	14	`$a0` = file descriptor,	bytes read (in `$v0`)
		`$a1` = buffer, `$a2` = count	
write	15	`$a0` = file descriptor,	bytes written (in `$v0`)
		`$a1` = buffer, `$a2` = count	
close	16	`$a0` = file descriptor	0 (in `$v0`)
exit2	17	`$a0` = value	

MANIPULATION OF REGISTERS

- Load addressing: Address of a variable is copied and stored in a temporary register.

  Example: la $t1, var1

- Indirect addressing: Value stored in a particular address is copied into a temporary register.

  Example: lw $t3, ($t0)

- Indexed addressing: Address of a register can be offset by a specified value to obtain a value stored in another address.

  Example: lw $t2, 3($t1)

SUBMISSIONS

- Submit your report and code via Canvas

- Your report must include:

1. Your name
2. A list of the assembly code file
3. A brief summary of project implementation
4. Results showing the working code via screen prints
5. The conclusion listing the lessons learned and problems faced