Binary/Decimal/Hexadecimal/ASCII Character Conversion Chart

bin	dec	hex	char
0	0	0	NUL
1	1	1	STX
10	2	2	SOT
11	3	3	ETX
100	4	4	EOT
101	5	5	ENQ
110	6	6	ACK
111	7	7	BEL
1000	8	8	BS
1001	9	9	HT
1010	10	A	LF
1011	11	B	VT
1100	12	C	FF
1101	13	D	CR
1110	14	E	SO
1111	15	F	SI
10000	16	10	DLE
10001	17	11	DC1
10010	18	12	DC2
10011	19	13	DC3
10100	20	14	DC4
10101	21	15	NAK
10110	22	16	SYN
10111	23	17	ETB
11000	24	18	CAN
11001	25	19	EM
11010	26	1A	SUB
11011	27	1B	ESC
11100	28	1C	FS
11101	29	1D	GS
11110	30	1E	RS
11111	31	1F	US
100000	32	20	SP
100001	33	21	!
100010	34	22	"
100011	35	23	#
100100	36	24	$
100101	37	25	%
100110	38	26	&
100111	39	27	'
101000	40	28	(
101001	41	29	)
101010	42	2A	*

bin	dec	hex	char
101011	43	2B	+
101100	44	2C	,
101101	45	2D	-
101110	46	2E	.
101111	47	2F	/
110000	48	30	0
110001	49	31	1
110010	50	32	2
110011	51	33	3
110100	52	34	4
110101	53	35	5
110110	54	36	6
110111	55	37	7
111000	56	38	8
111001	57	39	9
111010	58	3A	:
111011	59	3B	;
111100	60	3C	<
111101	61	3D	=
111110	62	3E	>
111111	63	3F	?
1000000	64	40	@
1000001	65	41	A
1000010	66	42	B
1000011	67	43	C
1000100	68	44	D
1000101	69	45	E
1000110	70	46	F
1000111	71	47	G
1001000	72	48	H
1001001	73	49	I
1001010	74	4A	J
1001011	75	4B	K
1001100	76	4C	L
1001101	77	4D	M
1001110	78	4E	N
1001111	79	4F	O
1010000	80	50	P
1010001	81	51	Q
1010010	82	52	R
1010011	83	53	S
1010100	84	54	T
1010101	85	55	U

bin	dec	hex	char
1010110	86	56	V
1010111	87	57	W
1011000	88	58	X
1011001	89	59	Y
1011010	90	5A	Z
1011011	91	5B	[
1011100	92	5C	\
1011101	93	5D	]
1011110	94	5E	^
1011111	95	5F	_
1100000	96	60	`
1100001	97	61	a
1100010	98	62	b
1100011	99	63	c
1100100	100	64	d
1100101	101	65	e
1100110	102	66	f
1100111	103	67	g
1101000	104	68	h
1101001	105	69	i
1101010	106	6A	j
1101011	107	6B	k
1101100	108	6C	l
1101101	109	6D	m
1101110	110	6E	n
1101111	111	6F	o
1110000	112	70	p
1110001	113	71	q
1110010	114	72	r
1110011	115	73	s
1110100	116	74	t
1110101	117	75	u
1110110	118	76	v
1110111	119	77	w
1111000	120	78	x
1111001	121	79	y
1111010	122	7A	z
1111011	123	7B	{
1111100	124	7C	|
1111101	125	7D	}
1111110	126	7E	~
1111111	127	7F	DEL

Please initial:

Please initial:

MIPS Reference Sheet

Basic Instruction Formats

	Register	`0000 00ss ssst tttt dddd d000 00ff ffff`	R	`s`, `t`, `d` are interpreted as unsigned
	Immediate	`oooo ooss ssst tttt iiii iiii iiii iiii`	I	`i` is interpreted as two's complement

Instructions

Word	.word i	`iiii iiii iiii iiii iiii iiii iiii iiii`		
Add	add $d, $s, $t	`0000 00ss ssst tttt dddd d000 0010 0000`	R	`$d = $s + $t`
Subtract	sub $d, $s, $t	`0000 00ss ssst tttt dddd d000 0010 0010`	R	`$d = $s - $t`
Multiply	mult $s, $t	`0000 00ss ssst tttt 0000 0000 0001 1000`	R	`hi:lo = $s * $t`
Multiply Unsigned	multu $s, $t	`0000 00ss ssst tttt 0000 0000 0001 1001`	R	`hi:lo = $s * $t`
Divide	div $s, $t	`0000 00ss ssst tttt 0000 0000 0001 1010`	R	`lo = $s / $t; hi = $s % $t`
Divide Unsigned	divu $s, $t	`0000 00ss ssst tttt 0000 0000 0001 1011`	R	`lo = $s / $t; hi = $s % $t`
Move From High/Remainder	mfhi $d	`0000 0000 0000 0000 dddd d000 0001 0000`	R	`$d = hi`
Move From Low/Quotient	mflo $d	`0000 0000 0000 0000 dddd d000 0001 0010`	R	`$d = lo`
Load Immediate And Skip	lis $d	`0000 0000 0000 0000 dddd d000 0001 0100`	R	`$d = MEM[pc]; pc = pc + 4`
Load Word	lw $t, i($s)	`1000 11ss ssst tttt iiii iiii iiii iiii`	I	`$t = MEM [$s + i]`
Store Word	sw $t, i($s)	`1010 11ss ssst tttt iiii iiii iiii iiii`	I	`MEM [$s + i] = $t`
Set Less Than	slt $d, $s, $t	`0000 00ss ssst tttt dddd d000 0010 1010`	R	`$d = 1 if $s < $t; 0 otherwise`
Set Less Than Unsigned	sltu $d, $s, $t	`0000 00ss ssst tttt dddd d000 0010 1011`	R	`$d = 1 if $s < $t; 0 otherwise`
Branch On Equal	beq $s, $t, i	`0001 00ss ssst tttt iiii iiii iiii iiii`	I	`if ($s == $t) pc += i * 4`
Branch On Not Equal	bne $s, $t, i	`0001 01ss ssst tttt iiii iiii iiii iiii`	I	`if ($s != $t) pc += i * 4`
Jump Register	jr $s	`0000 00ss sss0 0000 0000 0000 0000 1000`	R	`pc = $s`
Jump And Link Register	jalr $s	`0000 00ss sss0 0000 0000 0000 0000 1001`	R	`temp = $s; $31 = pc; pc = temp`

When a word is stored to memory location `0xffff000c`, the least-significant byte (eight bits) of the word are sent to the standard output.
Loading a word from memory location `0xffff0004` places the next byte from standard input into the least-significant byte of the destination register.

MIPS Executable Relocatable Linkable (MERL) Format

A .merl file is an executable MIPS binary file that is augmented by a table containing relocation and linking information. A .merl file has three components:

- **Header**: Three words consisting of
  - *Cookie*: a word containing 0x10000002 (which is the binary encoding of the MIPS instruction beq $0, $0, 2)
  - *Length*: the length (in bytes) of the .merl file
  - *CodeLength*: the length (in bytes) of the header plus the MIPS program (see below)
- **MIPS program**: a MIPS binary program encoded so as to execute correctly when loaded at RAM address 0xc (immediately following the header)
- **Relocation and External Symbol Table**: zero or more table entries, each having one of the following format:
  - *relocation entry*: each relocation entry contains two words:
    - *REL format code*: a word containing **0x01**
    - *location*: the location in the .merl file where the relocatable value is encoded
  - *external symbol definition*:
    - *ESD format code*: a word containing **0x05**
    - *value*: a 32-bit word encoding the (relocatable) value of the defined symbol
    - *name*: a 32-bit word encoding n, the number of characters in the symbol name, followed by n words, each encoding one of the characters in ASCII
  - *external symbol reference*: a relocatable value whose encoding is imported from some other .merl file, consisting of
    - *ESR format code*: a word containing **0x11**
    - *location*: the location in the .merl file where the value is to be encoded, once known
    - *name*: a 32-bit word encoding *n*, the number of characters in the symbol name, followed by *n* words, each encoding one of the characters in ASCII

A .merl file may be executed at location 0 as a .mips binary, or may be used as input to a loader or linker which creates a .mips binary to be executed at some other location, or which creates another .merl file.

Please initial:

CS 241 - WLP4 Programming Language Specification

Lexical Syntax

A WLP4 program is a sequence of *tokens* optionally separated by *white space* consisting of spaces, newlines, or comments. Every valid token is one of the following:

- ID: a string consisting of a letter (in the range a-z or A-Z) followed by zero or more letters and digits (in the range 0-9), but not equal to "wain", "int", "if", "else", "while", "println", "return", "NULL", "new" or "delete".
- NUM: a string consisting of a single digit (in the range 0-9) or two or more digits, the first of which is not 0; the numeric value of a NUM token cannot exceed 2^31-1
- LPAREN: the string "("
- RPAREN: the string ")"
- LBRACE: the string "{"
- RBRACE: the string "}"
- RETURN: the string "return" (in lower case)
- IF: the string "if"
- ELSE: the string "else"
- WHILE: the string "while"
- PRINTLN: the string "println"
- WAIN: the string "wain"
- BECOMES: the string "="
- INT: the string "int"
- EQ: the string "=="
- NE: the string "!="
- LT: the string "<"
- GT: the string ">"
- LE: the string "<="
- GE: the string ">="
- PLUS: the string "+"
- MINUS: the string "-"
- STAR: the string "*"
- SLASH: the string "/"
- PCT: the string "%"
- COMMA: the string ","
- SEMI: the string ";"
- NEW: the string "new"
- DELETE: the string "delete"
- LBRACK: the string "["
- RBRACK: the string "]"
- AMP: the string "&"
- NULL: the string "NULL"

White space consists of any sequence of the following:

- SPACE: (ascii 32)
- TAB: (ascii 9)
- NEWLINE: (ascii 10)
- COMMENT: the string "//" followed by all the characters up to and including the next NEWLINE

Any pair of consecutive tokens may be separated by white space. Pairs of consecutive tokens that both come from one of the following sets *must* be separated by white space:

- {ID, NUM, RETURN, IF, ELSE, WHILE, PRINTLN, WAIN, INT, NEW, NULL, DELETE}
- {EQ, NE, LT, LE, GT, GE, BECOMES}

Tokens that contain letters are case-sensitive; for example, `int` is an INT token, while `Int` is not.

Please initial:

Context-free Syntax

A context-free grammar for a valid WLP4 program is:

- terminal symbols: the set of valid tokens above
- nonterminal symbols: {procedures, procedure, main, params, paramlist, type, dcl, dcls, statements, lvalue, expr, statement, test, term, factor, arglist}
- start symbol: procedures
- production rules:

```
procedures → procedure procedures
procedures → main
procedure → INT ID LPAREN params RPAREN LBRACE dcls statements RETURN expr SEMI RBRACE
main → INT WAIN LPAREN dcl COMMA dcl RPAREN LBRACE dcls statements RETURN expr SEMI
RBRACE
params →
params → paramlist
paramlist → dcl
paramlist → dcl COMMA paramlist
type → INT
type → INT STAR
dcls →
dcls → dcls dcl BECOMES NUM SEMI
dcls → dcls dcl BECOMES NULL SEMI
dcl → type ID
statements →
statements → statements statement
statement → lvalue BECOMES expr SEMI
statement → IF LPAREN test RPAREN LBRACE statements RBRACE ELSE LBRACE statements RBRACE
statement → WHILE LPAREN test RPAREN LBRACE statements RBRACE
statement → PRINTLN LPAREN expr RPAREN SEMI
statement → DELETE LBRACK RBRACK expr SEMI
test → expr EQ expr
test → expr NE expr
test → expr LT expr
test → expr LE expr
test → expr GE expr
test → expr GT expr
expr → term
expr → expr PLUS term
expr → expr MINUS term
term → factor
term → term STAR factor
term → term SLASH factor
term → term PCT factor
factor → ID
factor → NUM
factor → NULL
factor → LPAREN expr RPAREN
factor → AMP lvalue
factor → STAR factor
factor → NEW INT LBRACK expr RBRACK
factor → ID LPAREN RPAREN
factor → ID LPAREN arglist RPAREN
arglist → expr
arglist → expr COMMA arglist
lvalue → ID
lvalue → STAR factor
lvalue → LPAREN lvalue RPAREN
```


Please initial:

Semantic Rules for WLP4

CS 241

Syntactic Categories

$E \in \{\text{expr}, \text{term}, \text{factor}, \text{lvalue}\}$ $\quad T \in \{\text{test}\}$ $\quad S \in \{\text{statement}, \text{statements}\}$ $\quad \tau \in \{\text{int}, \text{int}*\}$

Type Derivation Rules

[Literals and identifiers] $\dfrac{}{NUM : int}$ $\quad \dfrac{}{NULL : int*}$ $\quad \dfrac{\langle id, \tau \rangle \in \text{decls}}{id : \tau}$

[Parenthesized expressions] $\dfrac{E : \tau}{(E) : \tau}$

[Pointers] $\dfrac{E : int}{\&E : int*}$ $\quad \dfrac{E : int*}{*E : int}$ $\quad \dfrac{E : int}{\text{new int } [E] : int*}$

[Addition] $\dfrac{E_1 : int \quad E_2 : int}{E_1 \ + \ E_2 : int}$ $\quad \dfrac{E_1 : int* \quad E_2 : int}{E_1 \ + \ E_2 : int*}$ $\quad \dfrac{E_1 : int \quad E_2 : int*}{E_1 \ + \ E_2 : int*}$

[Subtraction] $\dfrac{E_1 : int \quad E_2 : int}{E_1 \ - \ E_2 : int}$ $\quad \dfrac{E_1 : int* \quad E_2 : int}{E_1 \ - \ E_2 : int*}$ $\quad \dfrac{E_1 : int* \quad E_2 : int*}{E_1 \ - \ E_2 : int}$

[Multiplication and division] $\dfrac{E_1 : int \quad E_2 : int}{E_1 \ * \ E_2 : int}$ $\quad \dfrac{E_1 : int \quad E_2 : int}{E_1 \ / \ E_2 : int}$ $\quad \dfrac{E_1 : int \quad E_2 : int}{E_1 \ \% \ E_2 : int}$

[Procedure calls] $\dfrac{\langle f, (\tau_1, \ldots \tau_n) \rangle \in \text{function-decls} \quad E_1 : \tau_1 \quad \cdots \quad E_n : \tau_n}{f(E_1, \ldots, E_n) : int}$

Please initial:

Type Correctness Rules

[Comparisons] $\dfrac{E_1 : \tau \quad E_2 : \tau}{\text{well-typed}(E_1 \texttt{ == } E_2)}$ $\dfrac{E_1 : \tau \quad E_2 : \tau}{\text{well-typed}(E_1 \texttt{ != } E_2)}$ $\dfrac{E_1 : \tau \quad E_2 : \tau}{\text{well-typed}(E_1 \texttt{ < } E_2)}$

$\dfrac{E_1 : \tau \quad E_2 : \tau}{\text{well-typed}(E_1 \texttt{ <= } E_2)}$ $\dfrac{E_1 : \tau \quad E_2 : \tau}{\text{well-typed}(E_1 \texttt{ > } E_2)}$ $\dfrac{E_1 : \tau \quad E_2 : \tau}{\text{well-typed}(E_1 \texttt{ >= } E_2)}$

[Control flow] $\dfrac{\text{well-typed}(T) \quad \text{well-typed}(S)}{\text{well-typed}(\texttt{while (}T\texttt{) \{}S\texttt{\}})}$ $\dfrac{\text{well-typed}(T) \quad \text{well-typed}(S_1) \quad \text{well-typed}(S_2)}{\text{well-typed}(\texttt{if (}T\texttt{) \{}S_1\texttt{\} else \{}S_2\texttt{\}})}$

[Deallocation] $\dfrac{E : \mathit{int}*}{\text{well-typed}(\texttt{delete [] }E\texttt{;})}$

[Printing] $\dfrac{E : \mathit{int}}{\text{well-typed}(\texttt{println }E\texttt{;})}$

[Assignment] $\dfrac{E_1 : \tau \quad E_2 : \tau}{\text{well-typed}(E_1 \texttt{ = } E_2\texttt{;})}$

[Sequencing] $\dfrac{}{\text{well-typed}(\varepsilon)}$ $\dfrac{\text{well-typed}(S_1) \quad \text{well-typed}(S_2)}{\text{well-typed}(S_1\ S_2)}$

[Decl'ns] $\dfrac{}{\text{well-typed}(\varepsilon)}$ $\dfrac{\text{well-typed}(\mathit{dcls})}{\text{well-typed}(\mathit{dcls}\texttt{ int }\mathit{id} = \mathit{NUM}\texttt{;})}$ $\dfrac{\text{well-typed}(\mathit{dcls})}{\text{well-typed}(\mathit{dcls}\texttt{ int* }\mathit{id} = \mathit{NULL}\texttt{;})}$

[Procedure] $\dfrac{\mathit{decl}_2 : \mathit{int} \quad \text{well-typed}(\mathit{decls}) \quad \text{well-typed}(S) \quad E : \mathit{int}}{\text{well-typed}(\texttt{int wain(}\mathit{decl}_1\texttt{, }\mathit{decl}_2\texttt{)\{}\mathit{decls}\ S\texttt{ return }E\texttt{;\}})}$

$\dfrac{\text{well-typed}(\mathit{decls}) \quad \text{well-typed}(S) \quad E : \mathit{int}}{\text{well-typed}(\texttt{int id(}\mathit{params}\texttt{)\{}\mathit{decls}\ S\texttt{ return }E\texttt{;\}})}$

Please initial: