Final Reference Sheet

Boolean Algebra

Law	AND Example	OR Example
Idempotent	A AND A = A	A OR A = A
Annulment	A AND 0 = 0	A OR 1 = 1
Identity	A AND 1 = A	A OR 0 = A
Complement	A AND (NOT A) = 0	A OR (NOT A) = 1
Double Complement	NOT(NOT A) = A	
Commutative	A AND B = B AND A	A OR B = B OR A
Distributive	A AND (B OR C) = (A AND B) OR (A AND C)	A OR (B AND C) = (A OR B) AND (A OR C)
Absorptive	A AND (A OR B) = A	A OR (A AND B) = A
Associative	(A AND B) AND C = A AND (B AND C)	(A OR B) OR C = A OR (B OR C)
De Morgan's Theorem	NOT(A AND B) = (NOT A) OR (NOT B)	NOT(A OR B) = (NOT A) AND (NOT B)

LC-3 Instruction Set

Opcode	Instruction	RTL Description	Format
0001	ADD	DR = SR1 + SR2	ADD DR, SR1, SR2
0001	ADD	DR = SR1 + imm5	ADD DR, SR1, imm5
0101	AND	DR = SR1 & SR2	AND DR, SR1, SR2
0101	AND	DR = SR1 & imm5	AND DR, SR1, imm5
0000	BR	if ((n & N) | (z & Z) | (p & P)) PC = PC + PCoffset9	BRnzp LABEL
1100	JMP	PC = BaseR	JMP BaseR
0100	JSR	R7 = PC; PC = PC + PCoffset11	JSR LABEL
0100	JSRR	R7 = PC; PC = BaseR	JSRR BaseR
0010	LD	DR = mem[PC + PCoffset9]	LD DR, LABEL
1010	LDI	DR = mem[mem[PC + PCoffset9]]	LDI DR, LABEL
0110	LDR	DR = mem[BaseR + offset6]	LDR DR, BaseR, offset6
1110	LEA	DR = PC + PCoffset9	LEA DR, LABEL
1001	NOT	DR = ~SR	NOT DR, SR
1101	RESERVED		
0011	ST	mem[PC + PCoffset9] = SR	ST SR, LABEL
1011	STI	mem[mem[PC + PCoffset9]] = SR	STI SR, LABEL
0111	STR	mem[BaseR + offset6] = SR	STR SR, BaseR, offset6
1111	TRAP	R7 = PC; PC = mem[TRAPVECT8]	TRAP TRAPVECT8

LC-3 Trap Routines

Trap Vector	Name	Description
x20	GETC	Read a single character from the keyboard. The character is not echoed onto the console. Its ASCII code is copied into R0. The high eight bits of R0 are cleared.
x21	OUT	Write a character in R0[7:0] to the console display.
x22	PUTS	Write a string of ASCII characters to the console display. The characters are contained in consecutive memory locations, one character per memory location, starting with the address specified in R0. The ASCII code contained in each memory location is copied to the console. The string is terminated with an ASCII NUL (x00) character.

Trap Vector	Name	Description
x23	IN	Print a prompt on the screen and read a single character from the keyboard. The character is echoed onto the console monitor, and its ASCII code is copied into R0. The high eight bits of R0 are cleared.
x24	PUTSP	Write a string of ASCII characters to the console. The characters are contained in consecutive memory locations, two characters per memory location, starting with the address specified in R0. The ASCII code contained in each memory location is copied to the console. The string is terminated with an ASCII NUL (x00) character.
x25	HALT	Halt execution and print a message on the console.

LC-3 Assembler Directives

Directive	Description
.ORIG	Tells the assembler where in memory to place the LC-3 program.
.FILL	Tells the assembler to set aside the next location in the program and initialize it with the value of the operand.
.BLKW	Tells the assembler to reserve the number of memory locations specified by the operand.
.STRINGZ	Tells the assembler to initialize memory with the ASCII codes corresponding to the characters in the string. The string is terminated with the ASCII NUL (x00) character.
.END	Tells the assembler that this is the end of the program.

LC-3 Notational Conventions

Directive	Description
xNumber	The number in hexadecimal notation. Example: xF2A1
#Number	The number in decimal notation. Example #793
bNumber	The number in binary. Example b10011
A[l:r]	The field delimited by bit [l] on the left and bit [r] on the right, of the datum A.
BaseR	Base Register; one of R0..R7, used with a six-bit offset to compute Base+offset addresses (LDR and STR), or to identify the address of a control instruction (JMP and JSRR).
DR	Destination Register; one of R0..R7, which specifies the register a result should be written to.
imm5	A five-bit immediate value, when used as a literal (immediate) value. Range: -16..15.
LABEL	A construct that identifies a location symbolically, by name, rather than its 16-bit address).
mem[address]	Denotes the contents of memory at the given address.
offset6	A six-bit signed 2's complement integer (bits [5:0] of an instruction), used with the Base+offset addressing mode. Bits [5:0] are sign-extended to 16 bits and added to the Base Register to form an address. Range: -32..31.
PC	Program Counter; 16-bit register that contains the memory address of the next instruction to be fetched.
PCoffset9	A nine-bit signed 2's complement integer (bits [8:0] of an instruction), used with the PC+offset addressing mode. Range -256..255.
PCoffset11	An eleven-bit signed 2's complement integer (bits [10:0] of an instruction), used with the JSR opcode to compute the target address of a subroutine call. Range -1024..1023.
setcc()	Indicates that condition codes N, Z, and P are set based on the value of the result written to DR.
SEXT(A)	Sign-extend A. The most significant bit of A is replicated to extend A to 16 bits.
SP	The current stack pointer. R6 is the current stack pointer.
SR, *SR1*, *SR2*	Source register; one of R0..R7 that specifies the register from which a source operand is obtained.
USP	The User Stack Pointer.
ZEXT(A)	Zero-extend A. Zeros are appended to the leftmost bit of A to extend it to 16 bits.

C Language

```
#include <stdio.h> // to include library files such as standard I/O (*stdio.h*)
#define STOP 0 // to set a value for a label
// comments code
int counter = 2; // integer, 12, 1400, 3
char PointLabel; // character 'c'
float temperature; // floating point 13.4, 1100.0
printf("%d\n", counter); // print a value or values
scanf("%d", &counter); // input a value or values
// formatting options
// ```%d``` - decimal integer
// ```%x``` - hexadecimal integer
// ```%c``` - ASCII character
// ```%f``` - floating-point number
NOT_A = !A; // logical NOT
A_and_B = A && B; // logical AND
A_or_B = A || B; // logical OR
if (a > b)  //  greater than
if (a >= b) //  greater than or equal
if (a < b)  //  less than
if (a <= b) //  less than or equal
if (a == b) //  equal
if (a != b) //  not equal
i++; // postincrement
i--; // postdecrement
++j; // preincrement
--j // predecrement

// if-else...
if (comparison)
{
    code;
}
else if (comparison)
{
    code;
}
else
{
    code;
}

// **While**
while (test)
  loop_body;

// **For**
for (init; end-test; step)
{
    statements;
}

// **Break and Continue**
break; // to exit the current loop
continue; // to continue iteration of the loop

// Pointers in C
int *p;  /* p is a pointer to an int */
value = *p; // returns the value pointed to by p
address = &z; //returns the address of variable z

// arrays
int variable[num_elements]; // declare array and size
variable[index]; // specific element

// Character String
char outputString[16];
char outputString[11] = "Result is ";
```

```
// Template for a C Program
#include <stdio.h>

// Subroutine
int function(int arg1, float arg2)
{
    // execute function using arguments
    return value;
}
// Function: main
// Description: counts down from user input to STOP
void main()
{
    // variable declarations

    // prompt user for input

    // perform task

    // print message

}
```

Functions and Arguments

C Function Argument Passing:

- **Arguments are passed by value** - functions receive **copies** of the values, not the original variables
- **Changes to parameters inside functions don't affect original arguments**
- **Pointers can be passed** to allow functions to modify original variables
- **Arrays are automatically passed as pointers** to their first element

Both function main and subfunctions must have a type indicating the *type* of returned value. This is indicated prior to the name of function, as in `int main()`. The word `void` indicates, no returned value.