CSE 560 **Summer 2009**
Charlie Giles

The "SummiX Machine" Characteristics

Distributed: Monday, June 22nd.

Memory

The machine's memory is word-addressable. That is, each 16-bit location is given a unique address. There are 65,536 words (i.e., 2^{16}) of memory, with addresses 0–65,535. Memory is organized into pages, each of size 512 (i.e., 2^9) words. There are 128 (i.e., 2^7) such pages.

Thus, a memory address is given by a 16-bit quantity where the upper 7 bits denote the page number and the lower 9 bits denote the offset within that page.

Registers

Program Counter The PC register is 16 bits. It contains the address of the next instruction to be fetched for execution (assuming that the current instruction is not a branch). Put another way, the value of the PC register is 1 greater than the address of the instruction currently executing.

General Purpose Registers There are 8 general purpose registers, each 16 bits. These registers are denoted R0, R1, ..., R7 and are used for computatation and indexing.

Condition Code Registers There are 3 condition code registers: N, Z, and P. These condition code registers are all updated (i.e., set or cleared) every time a value or result is written to a general purpose register (with one exception, noted later). The N register is set to 1 if and only if the last value written to the register was negative. Similarly, the Z (P) register is set to 1 if and only if the last value written was zero (positive). When there is no write to a general purpose register, these condition code registers are unchanged.

Arithmetic

Fixed-point only (i.e., no floating-point arithmetic). Negative numbers are represented in two's complement form. Overflows (positive and negative) can occur, for example x7FFF + x7FFF (i.e., $32,767 + 32,767$) yields xFFFE (i.e., -2).

Instruction Set Formats

Each instruction is a 16 bit (1 word) quantity. The four high-order bits (i.e., [15:12]) denote the opcode, while the rest encode the operands. The format of each instruction is shown in Figure 1.

An "x" indicates "don't care". The execution of the instruction does not depend on the value of this bit.

Data Processing Instructions

There are 3 instructions for processing data: ADD, AND, and NOT. All of these instructions modify the condition code registers.

The NOT instruction takes a single argument, contained in the source register SR, computes the bit-wise complement, and places the result in the destination register, DR.

The ADD and AND instructions each take two arguments and place their result in the destination register (DR). One of the arguments is contained in a source register (SR1) while the other can be in either a source register (SR2) or an immediate operand (imm5). In the case of immediate operands, the 5-bit quantity is sign extended to obtain a 16-bit operand.

Data Movement Instructions

There are seven instructions for moving data between memory and the registers: LD, LDR, LDI, LEA, ST, STR, and STI. The first four are load instructions that write to a general purpose register, while the rest are store instructions that write to memory. The load instructions modify the condition code registers while the store instructions do not.

The LEA (load effective address) instruction loads the destination register with the value obtained by concatenating bits [15:9] of the PC with bits [8:0] of the instruction (pgoffset9). This instruction uses *immediate addressing mode*.

The LD and ST instructions use *direct addressing mode*. The address of the operand is specified in the instruction. This address is formed from bits [15:9] of the PC with bits [8:0] of the instruction (pgoffset9).

The LDI and STI instructions use *indirect addressing mode*. An address is formed exactly as before with LD and ST. However, instead of this being the address of the operand to be loaded or stored, it contains the address of the operand to be loaded or stored. While the operand for LD and ST must be on the same page as the instruction, the operand for LDI and STI can be anywhere in memory.

The LDR and STR instructions use *register indexed addressing mode*. The address of the operand is formed by zero-extending the six bit offset (index6) and adding it to the value in the specified base register (BaseR). Thus, the index is always interpreted as a positive quantity (in the range 0-63).

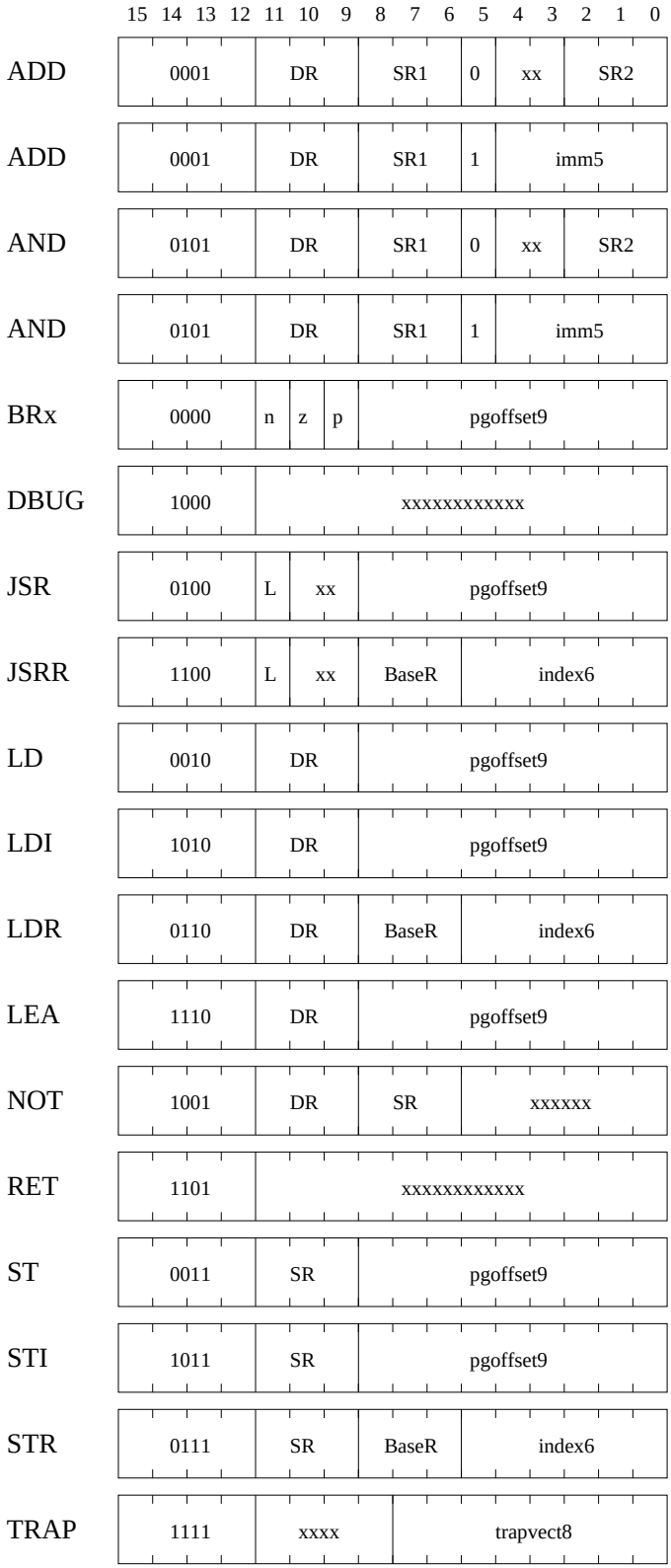

Figure 1: Formats of instructions

Flow of Control Instructions

There are five instructions for affecting flow of control: BRx, JSR, JSRR, RET, and TRAP. None of these instructions modify the condition code registers, except for possibly the TRAP instruction, as discussed below.

The BRx instruction is a conditional branch. The branch is taken if any of the specified condition code registers contain a 1. The destination of the branch is formed by concatenating bits [15:9] of the PC with bits [8:0] (pgoffset9) of the instruction. Note that 0000000000000000 is a no-op, while 0000111000000000 is an unconditional branch.

The TRAP instruction executes a system call. It branches to the address in the corresponding entry in the trap vector table (trapvect8). Your simulator should exhibit the following behavior according to the following trap vector table:

Number	Name	Description
x21	OUT	Write the character in R0[7:0] to the console.
x22	PUTS	Write the null-terminated string pointed to by R0 to the console.
x23	IN	Print a prompt on the screen and read a single character from the keyboard. The character is copied to the screen and its ASCII code is copied to R0. The high 8 bits of R0 are cleared.
x25	HALT	Halt execution and print a message to the console.
x31	OUTN	Write the value of R0 to the console as a decimal integer.
x33	INN	Print a prompt on the screen and read a decimal number from the keyboard. The number is echoed to the screen and stored in R0. You may place specific requirements on the size, formatting, etc. of the input.
x43	RND	Store a random number in R0.

(Note: The "x" before a number indicates that it is a hexadecimal number. Thus, x25 is 37 decimal. The instruction for halting the machine is 1111000000100101.)

When the trapvector is x23 or x33, the condition code registers are modified according to the value placed in R0. In addition to the behavior described above, a TRAP instruction also sets R7 to the PC. Thus, after completing a TRAP instruction, the value of R7 is equal to the address of the instruction following the TRAP.

The JSR and JSRR instructions allow jumps to subroutines. The PC is modified according to the operand of JSR/JSRR. The destination address for JSR is computed as with ST, while the destination address for JSRR is computed as with STR. If the link bit (L) is set, the value of PC is saved to R7 before branching. (If the L bit is not set, these instructions are written JMP and JMPR respectively.) Note that these instructions do not modify the condition code registers, despite updating a general purpose register (R7).

The RET instruction copies the contents of R7 to the PC. This instruction can be used to return from subroutine calls (assuming R7 has not been modified in the subroutine).

Miscellaneous

The DBUG instruction displays the contents of machine registers (PC, general purpose registers, condition code registers) to the console.