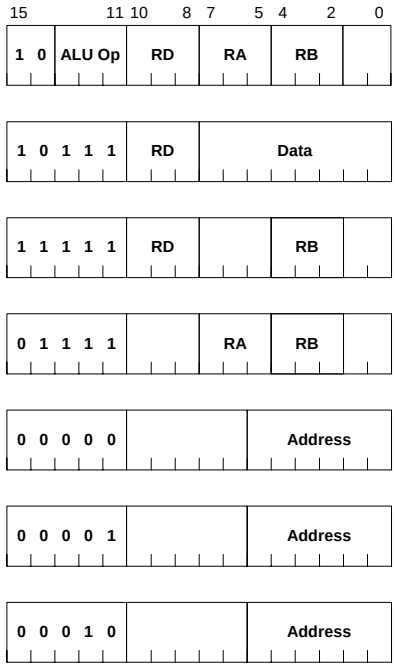
x370 Processor Definition

The x370 processor is a simple 16-bit architecture based on the ALU and register file we put together in class. The x370 has 4 registers and separate instruction and data memories. The instruction memory has 64 16-bit instructions and data memory has 256 16-bit data values. The instruction set has been defined to allow some expansion to the number of registers and the size of instruction memory. All x370 instructions can be executed in a single cycle.

Instruction Set



RD = RA op RB	ALU instruction
RD = Data	LDI - Load Immediate
RD = Dmem[RB]	LDR - Load Register
Dmem[RB] = RA	STR - Store Register
PC = Address	BR - Branch
if (Z) PC = Address	BRZ - Branch on Zero
if (N) PC = Address	BRN - Branch on Negative

ALU Instructions

ALU operations produce two status signals, Z and N, which indicate whether the result is Zero or Negative respectively. These signals are saved in a status register and can then be used by the branch instructions. The status register is latched only when an ALU instruction is executed (does not include LDI).

Name	Op code	Operation	Comments
ADD	10000	RD ← RA + RB	
XOR	10001	RD ← RA ⊕ RB	
INC	10010	RD ← RA + 1	
PASSA	10011	RD ← RA	
Reserved	10100		Available for new ALU operation
XNOR	10101	RD ← ¬(RA ⊕ RB)	
SUB	10110	RD ← RB – RA	Note order of operands
LDI	10111	RD ← Data (sign extended to 16 bits)	Non-ALU instruction: Load immediate loads data from instruction

Branch Instructions

Branch instructions allow the program to execute loops and execute different instructions depending on the result of an ALU operation. The conditional branch instructions test either the Z or N status signal saved in the status register. Branches are typically executed right after the ALU instruction that generates the desired status signal.

Name	Op code	Operation	Comments
BR	00000	PC ← Address	Unconditional branch
BRZ	00001	if (Z) PC ← Address; else PC ← PC + 1	
BRN	00010	if (N) PC ← Address; else PC ← PC + 1	

Load/Store Instructions

Data memory is accessed via the load and store instructions, which transfer a single value between a register and a location in data memory, whose address is given in register RB. Only the low-order 8 bits of RB is used for the address since the data memory has 256 locations.

Name	Op code	Operation	Comments
LDR	11111	RD ← DMEM[RB]	
STR	01111	DMEM[RB] ← RA	

Implementing the x370 Processor

We will implement the x370 in several steps instead of trying to do it all at once. This way, you can make sure it works as you go along.

Model 0 – ALU Instructions

The base processor is very simple – it executes ALU instructions only, starting after reset with the instruction at address 0, and then executing instructions at 1, 2, etc. We will give you this base processor. Your job is to add instructions and features to implement the full processor.

x370 Model 1 – Load Immediate Instruction (hand in Wednesday, Dec. 4)

Implement the load immediate (LDI) instruction, which allows the program to load a constant that is part of the instruction into the processor. This 8-bit constant is sign extended to allow negative constants.

x370 Model 2 – Branch Instructions (complete by Friday, Dec. 6)

The model 2 incorporates the branch instructions, which allows a program to execute loops and to branch based on the results of an ALU operation. There are three branch instructions: BR, unconditional branch,

BZ, branch if the ALU result was 0, and BN, branch if the ALU result was negative. The branch instructions specify the address of the next instruction to execute if the branch condition holds.

The ALU must be extended to produce two status signals: Z, which is true if the ALU result is 0, and N, if the result is <0. These signals must be saved until a branch instruction can use them. This is done by saving them in a status register, which is loaded whenever an ALU instruction is executed.

x370 Model 3 – Load/Store and Data Memory (complete by Wednesday Dec. 11)

So far, all the data used by the program is kept in the registers in the register file. To solve interesting problems, we need to have memory which contains input data and output data, as well as temporary data as needed. The Model 3 has a separate data memory with 256 locations. This memory is accessed via the LDR, Load Register, and STR, Store Register, instructions. In both cases, the address of the location in data memory is given by register RB. The LDR instruction loads this memory location into RD, and the STR instruction stores RA to this memory location.

The data memory is implemented using the dram.v module. This module has a parameter that gives the name of the file that is used to initialize the memory contents. You can change the file name by right-clicking on the dram module and changing this parameter.

The Final Program (hand in electronically, Friday Dec. 13)

You will write a program that runs on the x370 that does a "phone book search". We will give you a list of 120 pairs of numbers, each representing a name and a phone number pair. You may save this in the data memory however you like by defining your own "dram.dat" initial data file. Your program will then search for a "name" in this set of data, and return the corresponding "phone number". We will use location 0 of the data memory as the "name" to search for. Your program should store the associated phone number in data memory location 1 and then halt by branching and looping at location 63 (the last instruction). You must turn in your project electronically and we will run your program to test it.

Extra Credit (hand in electronically as Final Program, Friday Dec. 13)

As an extra challenge, you may be interested is seeing how fast you can get your x370 to execute this task. You may choose many different ways to speed your implementation:

1) Faster processor design and thus a faster clock rate
2) Improved architecture: e.g. more registers and/or new instructions
3) Fast algorithms. A simple linear search is fine as a start, but there are much faster ways to search for data.

To time your implementation, we will implement "memory-mapped" I/O. To do this, we will make address 255 (11111111) special. The data memory will not respond to this address. Instead, a special I/O module will return a value if this address is read, and will save a value if this address is written. Your program will read "key" values from this location, look up the corresponding phone number and write the result to this location. The I/O module will give a new value on every read, and will store the values written to it. This module will check the validity of the results, and will print out the elapsed time once all lookups have been performed. More details on how this interface works will be published later.