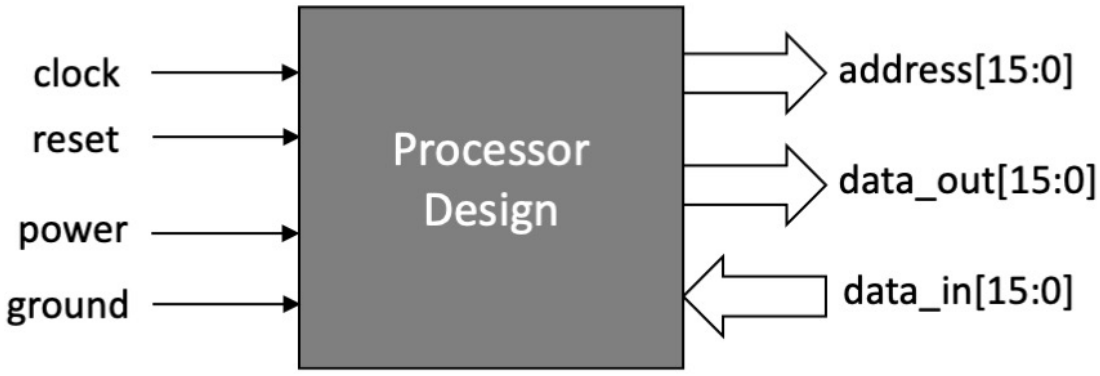
COMP22111: Processor Microarchitecture

The LUMP Processor – an Overview

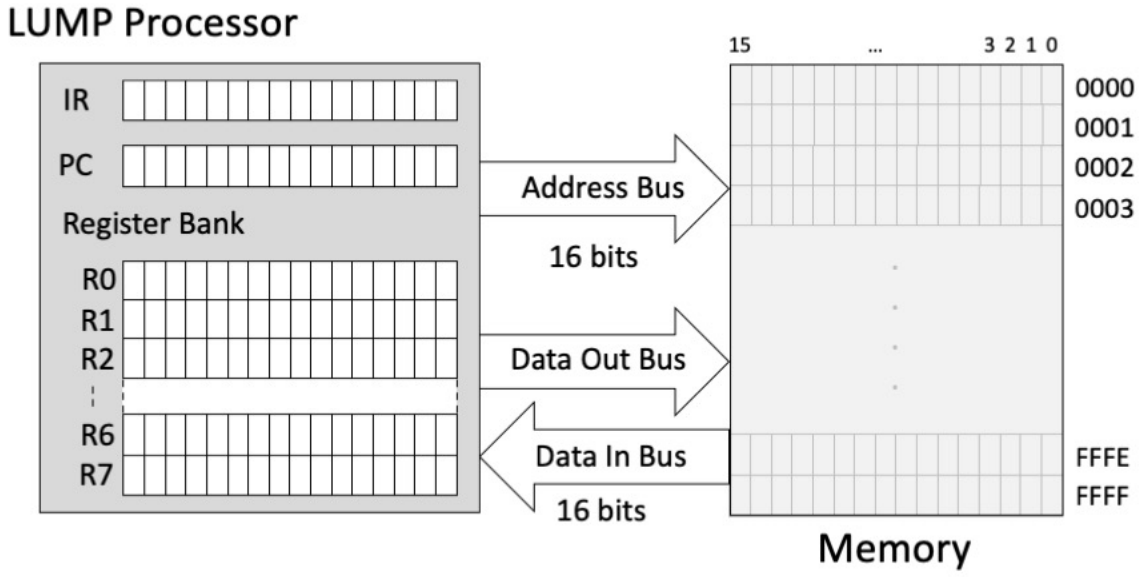
The LUMP processor, or Limited Use Made-up Processor, is a very limited processor design created purely as a design example beyond MU0 (which has several significant differences to Stump) but bears some relationship to the Stump processor.

LUMP is a 16-bit processor and, like Stump, is a RISC design, so it has a load/store architecture meaning that operations are only performed on data stored in a local register bank. Separate load and store instructions are provided to support data move to and from main memory.

The design has a 16-bit address bus to memory and separate 16-bit data out and data in buses for the movement of data to and from memory.



To support the load/store architecture, the LUMP processor contains a register bank with 8 registers number R0 to R7. However, unlike the Stump processor the Program Counter is not part of the register and is a separate register along with the Instruction Register.



LUMP Instruction Format

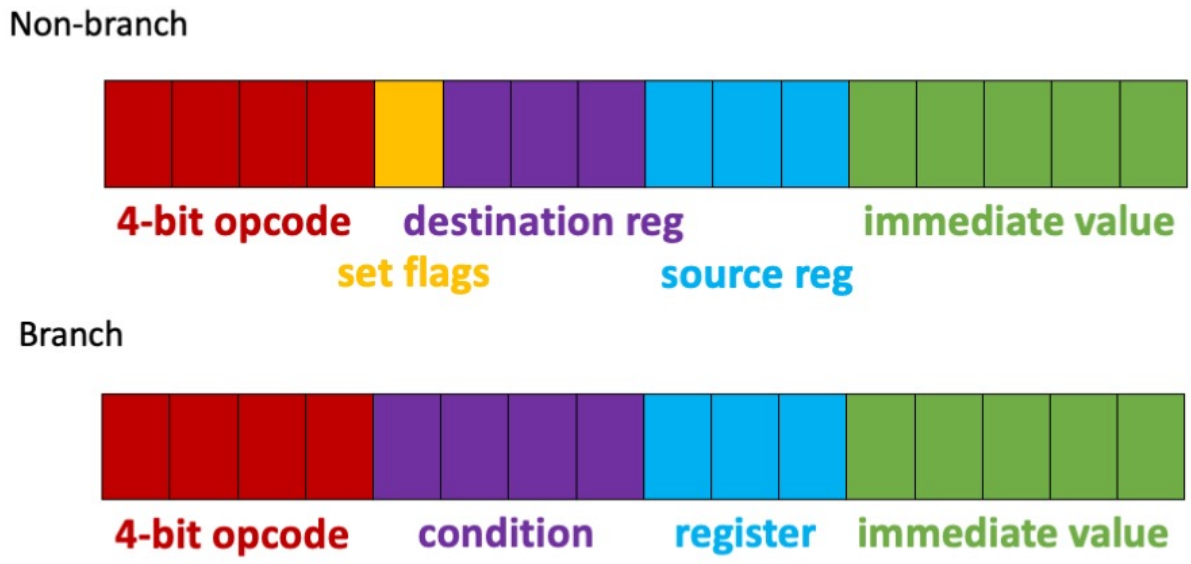
There are two instruction formats for LUMP as shown diagrammatically below. First, we have non-conditional branch instructions, the format of these instructions are as follows:

- bits 15:12: 4-bit opcode that specifies the instruction
- bit 11: indicates if flags are to be set of not. If '1' then flags are set, if '0' flags are not set. Like ARM & Stump this is indicated in the instruction by appending an S.
- bits 10:8: 3-bit field that specifies the destination register in the register bank, so 000 is R0, 001 is R1 etc
- bits 7:5: 3-bit field that specifies a register in the register bank that holds one of the operands
- bits 4:0: 5-bit immediate value that specifies the second operand.

We also have a separate branch instruction:

- bits 15:12: again, forms the opcode, which is the same for all branch instructions
- bits 11:8: 4-bit field that specifies the condition, giving 16 possible conditional branch instructions.
- bits 7:5: 3-bit field that specifies the source register for part of the branch address.
- bits 4:0: 5-bit field that forms an immediate value that forms part of the branch address.

In the case of LUMP, the branch address is formed from adding an offset to the PC. The 3-bit register field is currently ignored.



LUMP Instructions

The range of LUMP instructions are illustrated in the table given below. We have two arithmetic operations: ADD, SUB; three logical operations: AND, OR, XOR; two data movement operations: LD and ST; and the conditional branch instructions Bcc. As there are

only 8 instructions specified and the opcode is a 4-bit value, then there are 8 unused instructions, which allow for future expansion.

Mnemonic	INSTR	Operation
ADD	0000	2's complement ADD
SUB	0010	2's complement SUB
AND	0100	Bitwise AND of two 16-bit words
OR	0101	Bitwise OR of two 16-bit words
XOR	0110	Bitwise XOR of two 16-bit words
LD	1000	Load operation
ST	1001	Store operation
Bcc	1111	Branch taken if condition code cc is satisfied

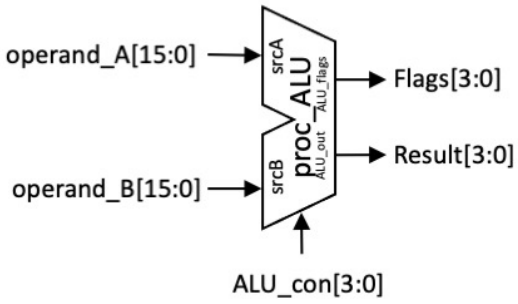
When it comes to conditional branch instructions, LUMP has a similar range compared to STUMP.

Bcc	Code
BAL	0000
BNV	0001
BHI	0010
BLS	0011
BCC	0100
BCS	0101
BNE	0110
BEQ	0111
BVC	1000
BVS	1001
BPL	1010
BMI	1011
BGE	1100
BLT	1101
BGT	1110
BLE	1111

LUMP ALU

The arithmetic logic unit (ALU) in LUMP must support the instructions provided. Consequently, a 4-bit control signal is used to select the required operation, as shown below. As only 5 operations are currently required, then several states of the control signal are unused at present. However, these available for future expansion.

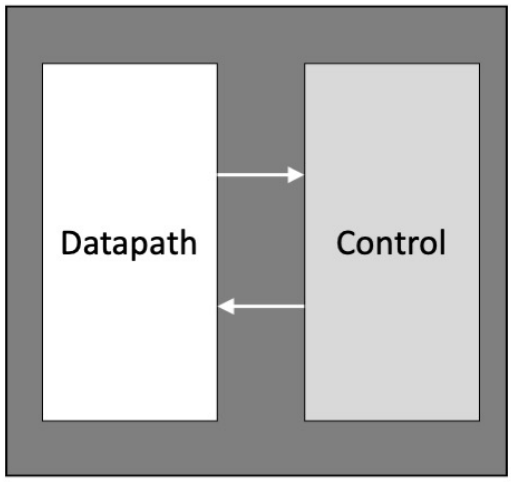
Please note: LUMP does not use the ALU to perform the PC update, as is the case with Stump. Instead, a separate block of logic generates PC+1. This means the ALU is not used during fetch meaning that pipelining of the processor is possible, to speed up performance.



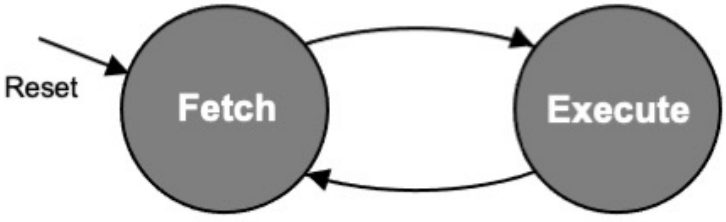
ALU_con	Operation
0000	operand_A + operand_B
0001	operand_A – operand_B
0010	operand_A & operand_B
0011	operand_A | operand_B
0100	operand_A ⊕ operand_B
0101 - 1111	Unused

LUMP Control

LUMP is a sequential system just like any processor. As such, we can split the design of the processor into datapath and control.

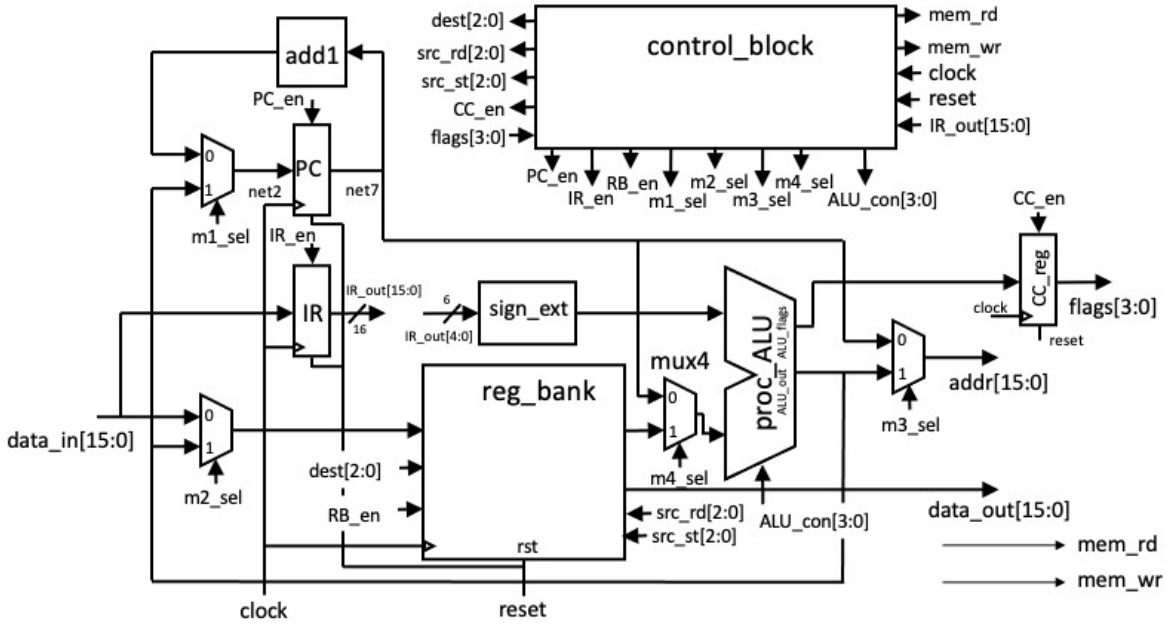


In the case of the control, the finite state machine for LUMP is very simple, it has 2 states: fetch and execute.



LUMP Datapath

One design for the LUMP datapath, along with the required control signals from the control, is shown below.



We won't go into the detail of where this design came from (apart from in my head) but here are some key points:

- the data to the register bank (reg_bank) comes from either the data_in bus, to support the load instruction, or from the output of the ALU (proc_ALU), to support register writeback – this is controlled by mux2.
- in the case of store, there is a dedicated output from the reg_bank connected to the data_out bus.
- the ALU takes data from the reg_bank and the IR, which is sign extended to 16 bits.
- the data into the PC comes from the output of an incrementer unit (add1) – the input of which comes from the output of the PC, for the fetch increment, or from the output of the ALU for a branch; this is controlled by mux1.
- the address supplied to the address bus, addr, comes from either the ALU, for load/store instructions, or from the PC, for fetch.
- all multiplexers are 2-to-1 16-bit multiplexers
- all registers are 16-bit registers with clock and reset signals omitted for clarity
- in total, the LUMP datapath will have 9 functional components:
    - one register bank called reg_bank
    - one ALU called proc_ALU
    - four multiplexers called mux1, mux2, mux3, mux4
    - three registers called PC, IR and CC_reg
    - an increment block called add1, which is a combinatorial circuit is simply the input + 1

- a sign extender block called sign_ext, which sign extends the 6-bit input to 16-bits.

We will use LUMP as a design example in lectures when we go through the various steps of implementation.