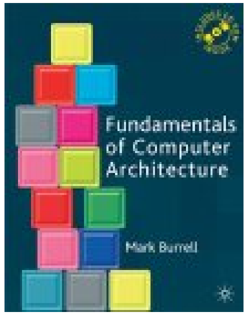
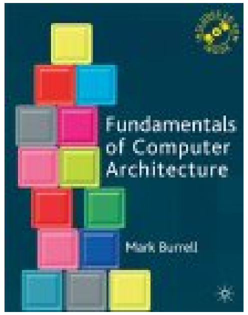


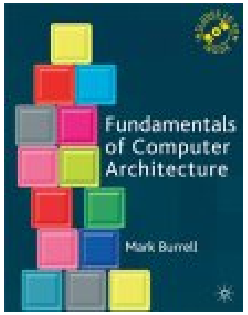
Fundamentals of Computer Architecture

8. Bringing It All Together – The Hardware Engineer's Perspective



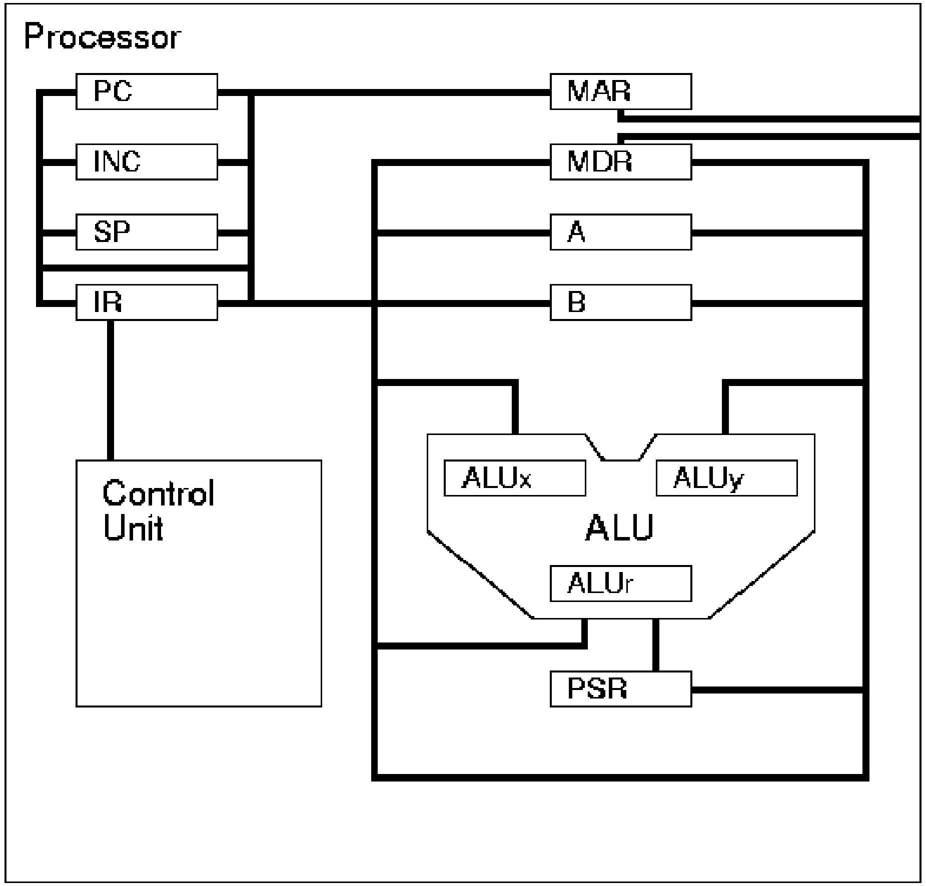
CHAPTER OVERVIEW

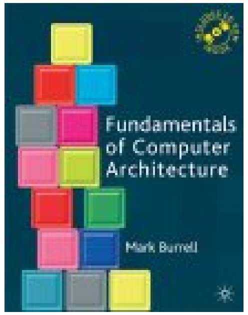
- This chapter includes:
  - Assigning tasks to individual processor components;
  - Micro-instructions;
  - Instruction sets;
  - The format of a program;
  - The fetch-execute cycle;
  - Executing programs in JASPer.



Recap

- So what sort of tasks does the processor have to do?
  - It needs to be able to read bit patterns from, and write bit patterns to, memory. We saw how it could do this in the previous chapter;
  - It needs to keep track of where we are within a program, so it knows what to do next;
  - It needs to know what to do for a particular instruction;
  - It needs to be able to perform arithmetic and logical operations;
  - It needs some general purpose storage areas.

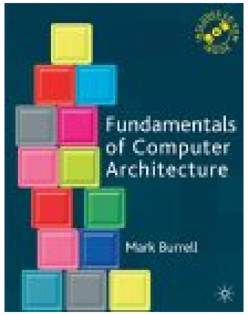




Introducing Micro-Instructions

- *Data movement micro-instructions*
  - MAR ← [A];
- *ALU micro-instructions*
  - ALUr = [ALUx]+[ALUy]
- *Test micro-instructions*
  - if(PSR(z) == 1)
- *Processor control micro-instructions*
  - halt

- How can we go about obtaining the logical AND of the bit patterns stored in the A and B registers?
  - ALUx ← [A]
  - ALUy ← [B]
  - ALUr = [ALUx]&[ALUy]
  - A ← [ALUr]



Introducing The Instruction Set

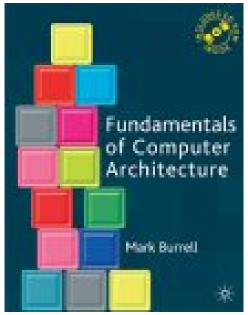
Comments

Directive

```
*
* Our first sum program
*
org 0
9005        MOVE #$05,A      * Transfer the first data value
                             *   to the A register
9103        MOVE #$03,B      * Transfer the second data value
                             *   the B register
0600        ADD  B,A         * Add them, storing the result i
                             *   the A register
F000        HALT             * Stop the program
```

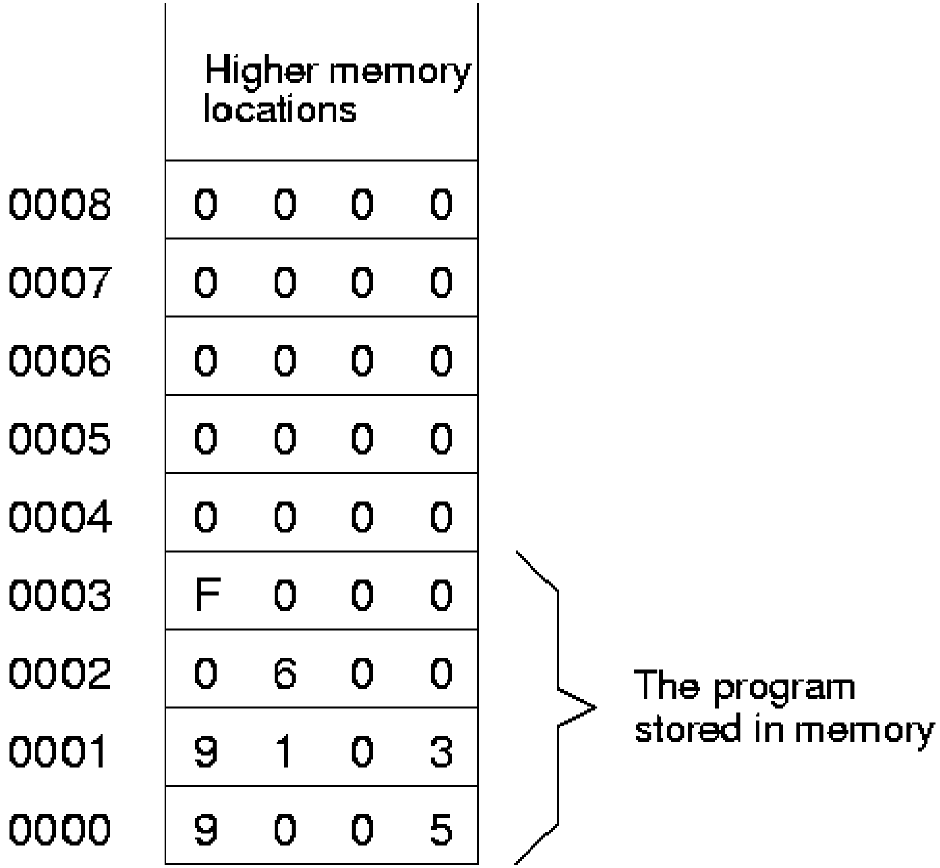
Machine code | Mnemonics and operands | Comments

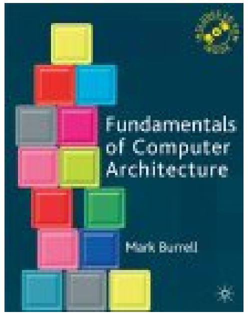
- If we were to write programs using micro-instructions
  - they would be extraordinarily long and very difficult to check that we haven’t introduced any mistakes.
  - Therefore, we don’t.
- Instead, we group sets of micro-instructions together to form higher level instructions, known as *assembly language* instructions.
  - AND B,A



Our First Program

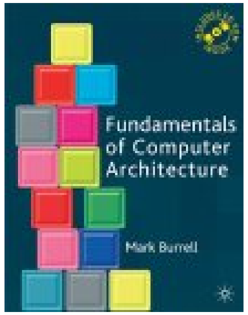
- Our program in memory – compare with the previous slide





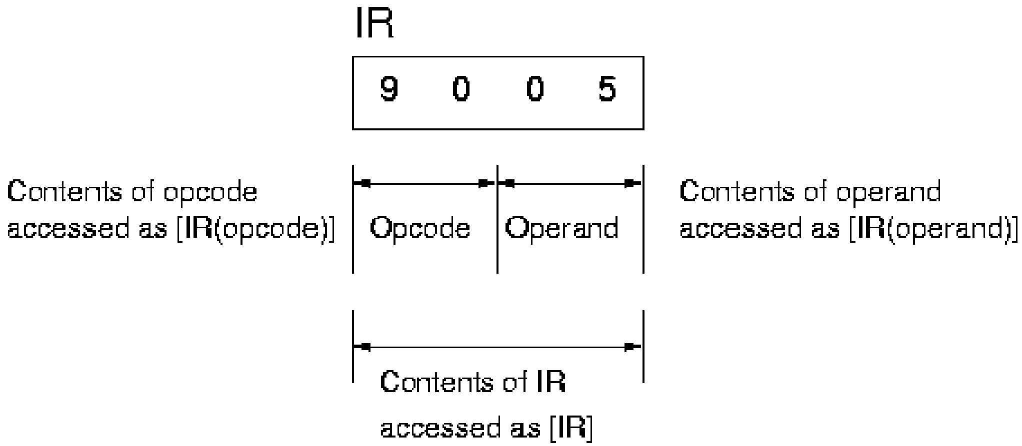
The Fetch-Execute Cycle

- Running the program
  - The processor loads, or *fetches*, the first instruction from memory (stored in memory at location $0000) into the IR;
  - Next it runs, or *executes*, this first instruction - it is the CU that does this, and as already seen, the A register takes the bit pattern $05;
  - It then fetches the second instruction from memory and stores it in the IR;
  - Next it executes this second instruction - the B register takes the bit pattern $03;
  - etc.
- The only ways that the processor would cease to fetch the next instruction are:
  - If the power to the processor is switched off;
  - The halt microcode is executed;
  - The processor reset button is pressed.



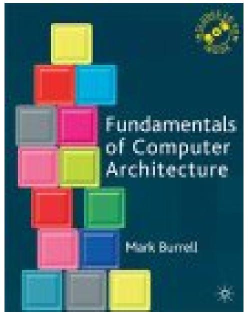
Inside The Fetch-Execute Cycle

- In our simple processor, the fetch cycle is defined as the following RTL sequence:
  - 1 MAR ← [PC]
  - 2 INC ← [PC]
  - 3 PC ← [INC]
  - 4 MDR ← [M[MAR]]
  - 5 IR ← [MDR]
  - 6 CU ← [IR(opcode)]



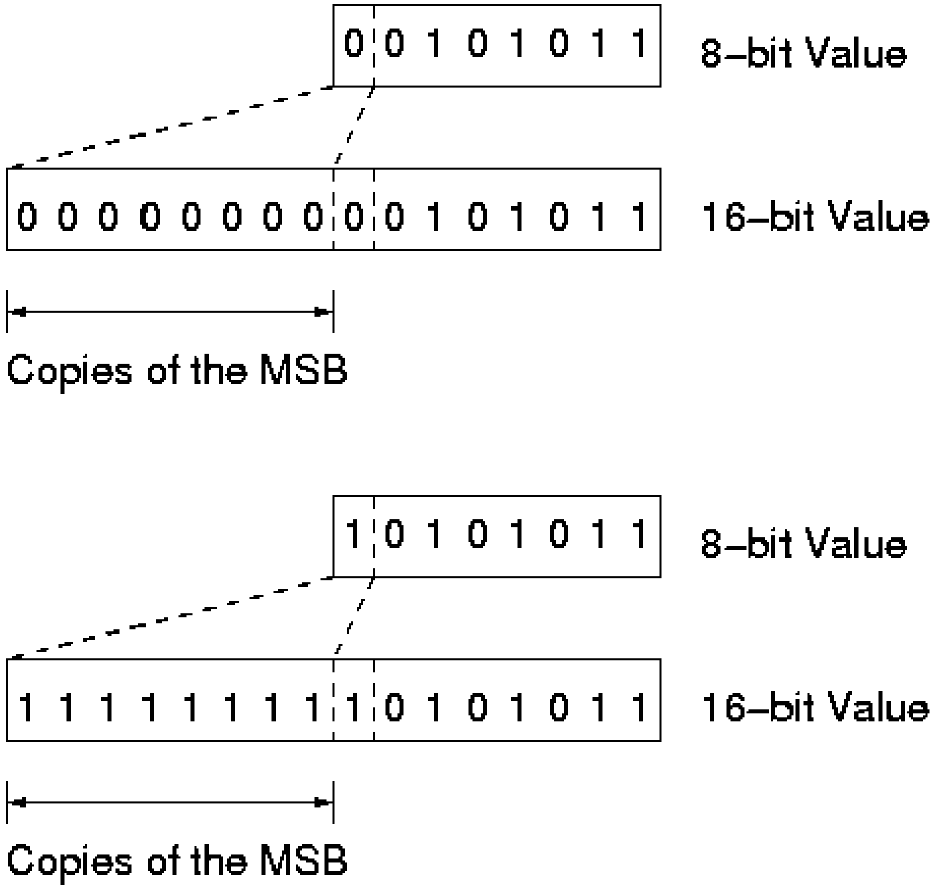
The execute cycle for the given opcode:

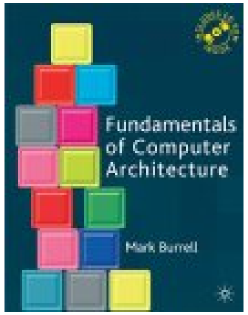
- 1 ALUx ← [A]
- 2 ALUy ← [B]
- 3 ALUr = [ALUx] + [ALUy]
- 4 A ← [ALUr]



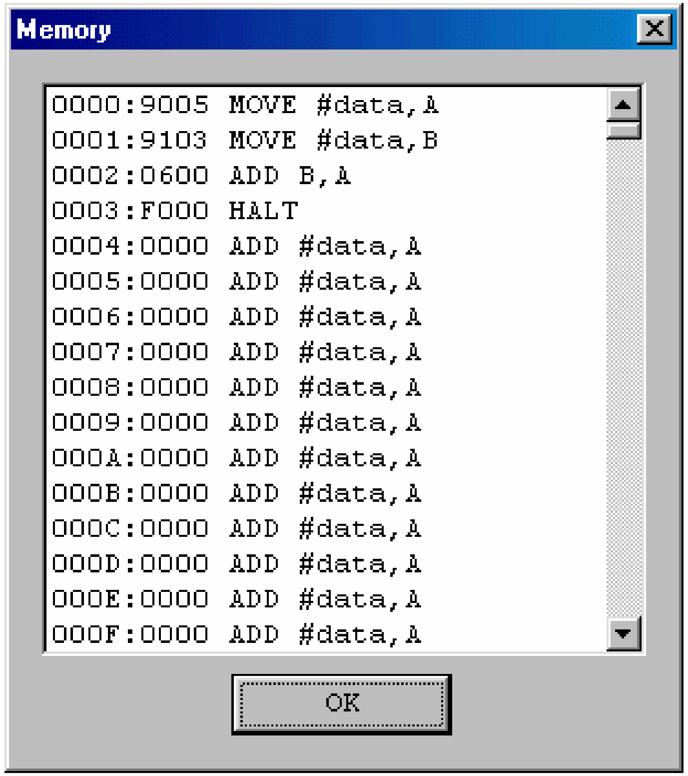
Sign Extension

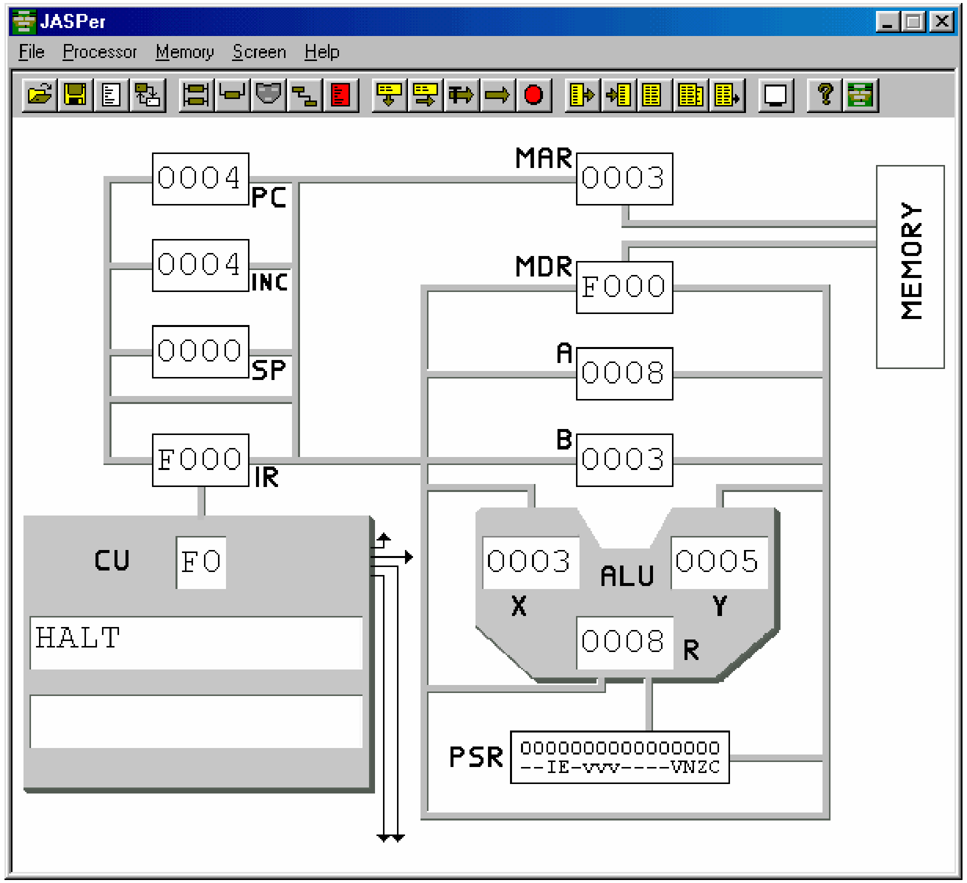
- When we perform a data movement from the IR(operand) the data value is *sign extended* to 16 bits

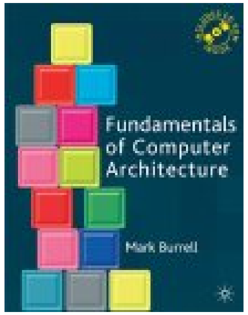




Running The Program In JASPer

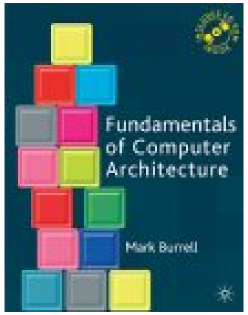






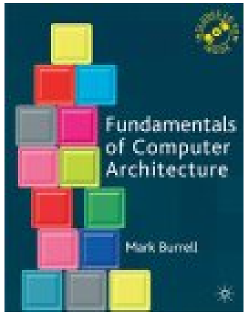
Chapter Summary

– **Tasks for individual registers**
  - The PC is used to bookmark which instruction the processor is to execute next;
  - The INC is used to add 1 to the PC;
  - The MAR and MDR are used to access memory;
  - A and B are general purpose registers;
  - The IR is used to store the instruction;
  - The ALUx and ALUy are the ALU inputs;
  - The ALUr is the ALU output, and the PSR contains flags that are updated by the ALU.



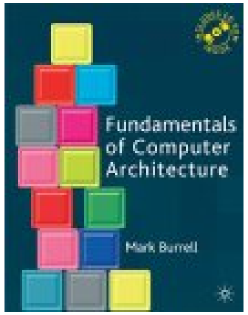
Chapter Summary

- **Micro-instructions**
  – The four sets of micro-instructions understood by our simple processor are
    - the data movement micro-instructions,
    - the ALU micro-instructions,
    - the test micro-instructions
    - and the control micro-instructions;
  – All micro-instructions can be represented by an RTL description;
  – Assembly language instructions can be defined using micro-instructions.



Chapter Summary

- **Instruction sets**
  - The number of instructions within an instruction set is limited by the width of the opcode;
  - Our simple processor has an opcode width of eight bits and therefore we can have a maximum of 256 instructions in an instruction set. In reality we do not need this many to write useful programs.



Chapter Summary

- **The format of a program**
  - The processor executes the machine codes of a program;
  - Additionally we add mnemonics, operands and comments so that we can understand what the individual machine codes are to do.
- **The fetch-execute cycle**
  - The processor runs programs by using the fetch-execute cycle;
  - Each instruction in memory is in turn, fetched, placed in the IR, and then executed by the CU.