Chapter 04: Instruction Sets and the Processor organizations

Lesson 18:

Stack-based processor Organisation

Objective

- To understand stack based processor organisation
- Instruction set of a stack organized processor
- Learn Post Fix Notations for Stack based Computers
- Learn Difference in Instructions encoding of Stack based and GPR based

Stack based organisation

Stack-based organization

1. First, because the operands and destination of an instruction in a stack-based organization are implicit, instructions take fewer bits to encode than they do in general-purpose register organizations
- This reduced the amount of memory taken up by programs, which was a significant issue on early machines

Stack-based organization

2. Second, stack-based architectures manage registers automatically, freeing programmers from the need to decide which data should be kept in the register file

Stack-based organization

3. Advantage of stack-based organizations is that the instruction set does not change if the sizes of the register file changes

- This means that programs written for a stack-based processor can be run on future versions of the processor that have more registers, and they will see performance improvements because they are able to keep more of the stack in registers

Stack-based organization

4. Stack-based organizations are also very easy to compile to—so easy that some compilers generate stack-based versions of a program as part of the compilation process even when compiling for a GPR processor

Stack-based organization

- In a, the register file invisible to the program
- Instructions read their operands from, and write their results to, a *stack*, and it has last-in-first-out (LIFO) data-structure
- Stack-based organizations use the stack in place of random access register file (GPRs) for many instructions

Advantages

- Fewer bits for the instruction as the operands and destination of an instruction are implicit
- Automatic management of stacked registers

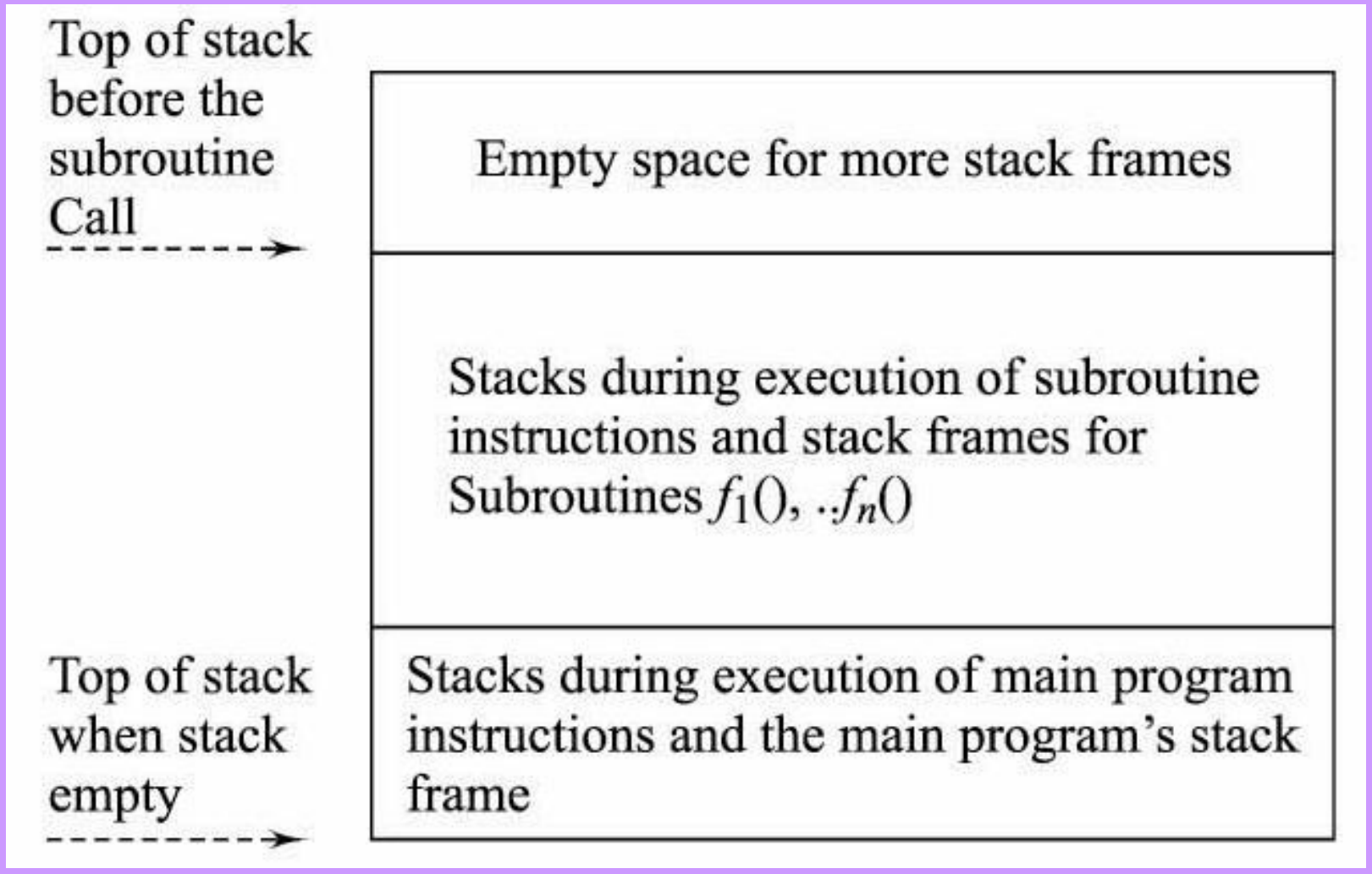
Advantages

- No change in instruction set if the sizes of the stacked register file changes
- Easy compilation
- Programs take up very little memory, because it is not necessary to specify where the source and destination of the operations are located

Stack-based and general-purpose register organizations differ

- Mainly in their interfaces to their register files
- In stack-based organizations, data is stored on a stack in memory
- The processor's file may be used to implement the top portion of the stack to allow faster access to the stack

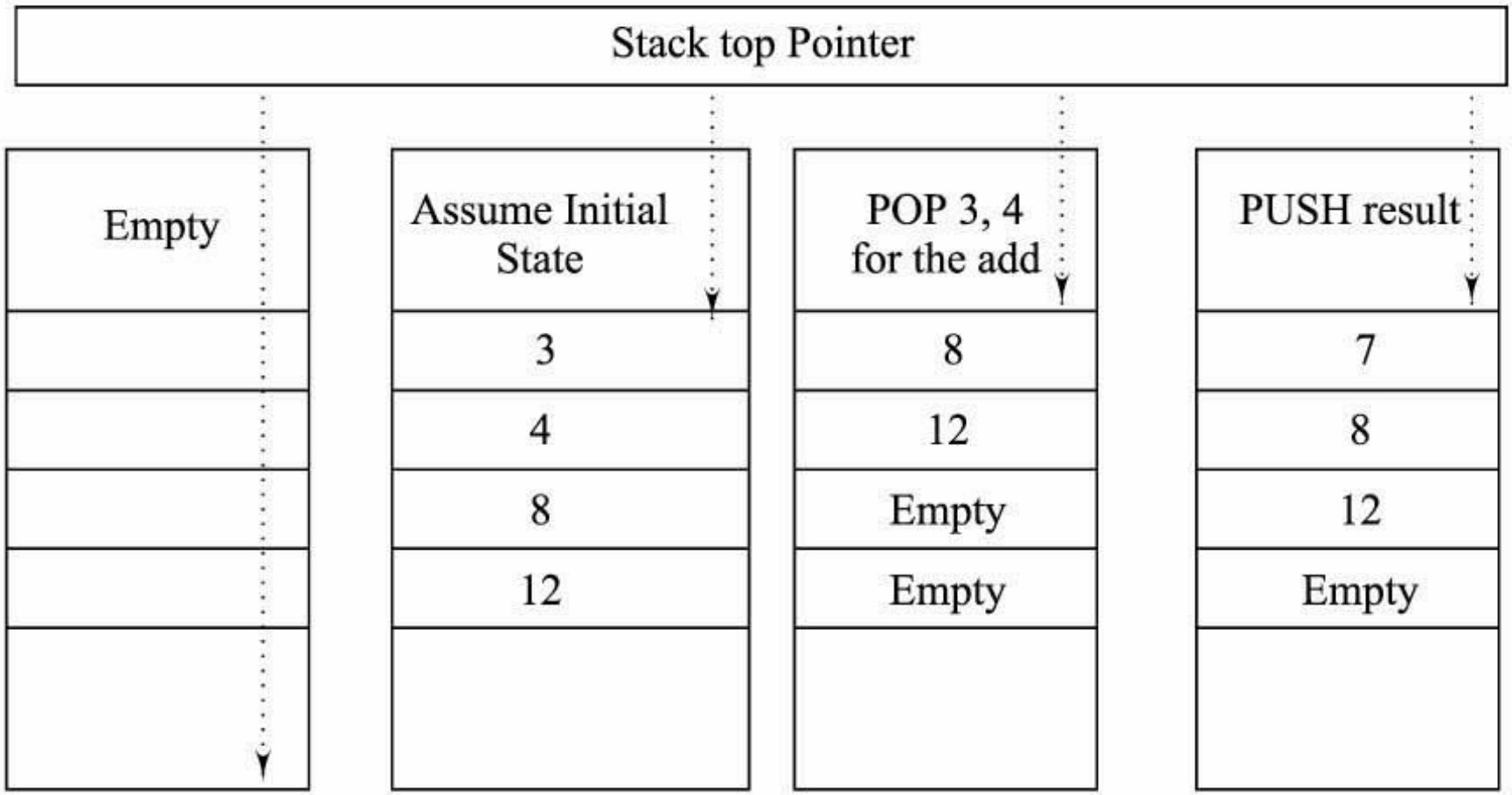
Processor stack in stack organised computer



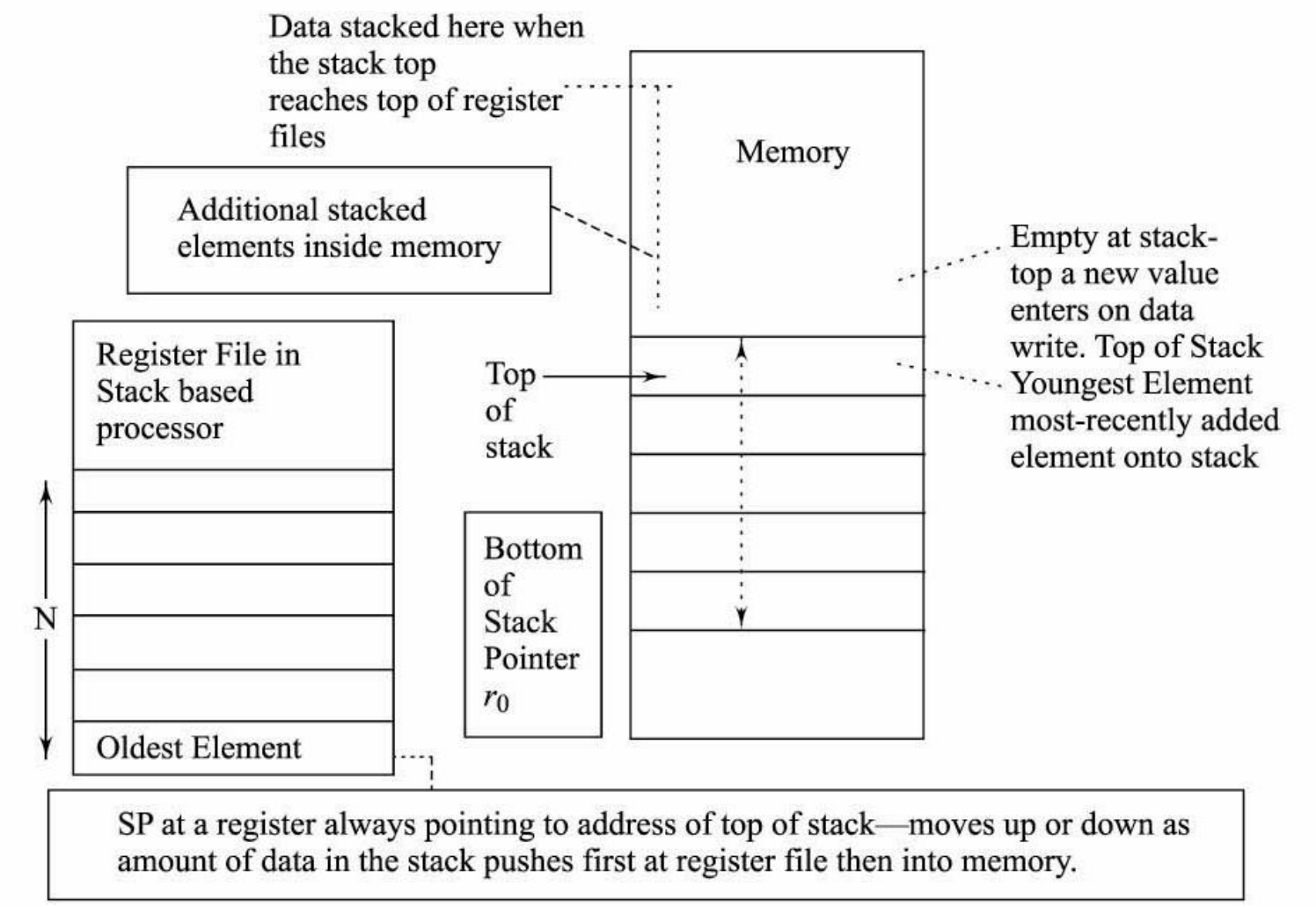
Stack-based organization Operands

- Get their operands from and write their results to the stack
- When an instruction executes, it pops its operands off of the stack, performs the required computation, and pushes the result onto the top of the stack

Stack based execution of instructions



Speed up of Stack access by a register file invisible to programmer



Embedded systems

- Stack-based organizations are somewhat more attractive for embedded systems, in which low cost and low power consumption requirements often limit the amount of memory that can be included in a system, making code size more of concern

Instruction set of a Stack based organisation Processor

Stack Organised Instruction Set

- PUSH #X STACK ← X
- POP a ← STACK
- LD a ← STACK; STACK ← Address of (a)
- ST a ← STACK Address of (a) ← STACK;

Stack Organised Instruction Set

- ADD a ← STACK; b ← STACK; STACK ←a + b Integers
- FADD a ← STACK; b ← STACK; STACK ←a + b Floating Points
- SUB a ← STACK; b ← STACK; STACK ←a – b Integers
- FSUB a ← STACK; b ← STACK; STACK ←a – b Floating Points

Stack Organised Instruction Set

- MUL a ← STACK; b ← STACK; STACK ←a × b Integers
- FMUL a ← STACK; b ← STACK; STACK ←a × b Floating Points
- DIV a ← STACK; b ← STACK; STACK ←a ÷ b Integers
- FDIV a ← STACK; b ← STACK; STACK ←a ÷ b Floating Points

Stack Organised Instruction Set

- AND a ← STACK; b ← STACK; STACK ←a .AND. b Binary Numbers
- OR a ← STACK; b ← STACK; STACK ←a .OR. b Binary Numbers
- NOT a ← STACK; STACK ← **NOT** Binary Numbers

Stack Organised Instruction Set

- ASR a ← STACK; b ← STACK; STACK ←a + b arithmetic left or right (+ or –) shift by b
- LSH a ← STACK; b ← STACK; STACK ←a + b logical left or right (+ or –) shift by b
- BR PC ← STACK;
- BEQ a ← STACK; b ← STACK; c ← STACK; PC ← c if b = a

Stack Organised Instruction Set

- BNE a ← STACK; b ← STACK; c ← STACK; PC ← c if b ≠ a
- BLT a ← STACK; b ← STACK; c ← STACK; PC ← c if b < a
- BGT a ← STACK; b ← STACK; c ← STACK; PC ← c if b > a
- BLE a ← STACK; b ← STACK; c ← STACK; PC ← c if b is less or equal to a

Stack Organised Instruction Set

- BGE a ← STACK; b ← STACK; c ← STACK; PC ← c if b is greater or equal to a
- BGE Label rb, rc PC ← label assigned bits by program text/assembler if rb > or = rc

Encoding in Postfix Notations

Stack-based programs

- Simply sequences of instructions that get executed one after the other
- Given a stack-based program and an initial configuration of the stack, the result of the program can be computed by applying the first instruction in the program, determining the state of the stack and memory after the first instruction completes, and repeating with each successive instruction

Writing programs for stack-based processors

- Somewhat more difficult, because stack-based processors are better suited to postfix notation (RPN) [Reversed Polish Notation] than traditional infix notation
- Infix notation is the traditional way of representing mathematical expressions, in which the operation is placed between the operands

Postfix notation

- The operation is placed after the operands. For example, the infix expression 2 +3 becomes 2 3 + in postfix notation. Once an expression has been recoded in postfix notation, converting it into a stack-based program is straightforward.

Postfix notation

- Starting at the left, each constant is replaced with a PUSH operation to place the constant on the stack, and operators are replaced with the appropriate instruction to perform their operation

Example of a stack-based program for 2 + (7 × 3)

- Convert the expression into postfix notation
- Best done by iteratively converting each sub-expression into postfix, so 2 + (7 ×3) becomes 2 + (7 3 ×) and then 2 (7 3 ×) +

Conversion of t the postfix expression into a series of instructions

- PUSH #2
- PUSH #7
- PUSH #3
- MUL
- ADD

Verify the this program correctness

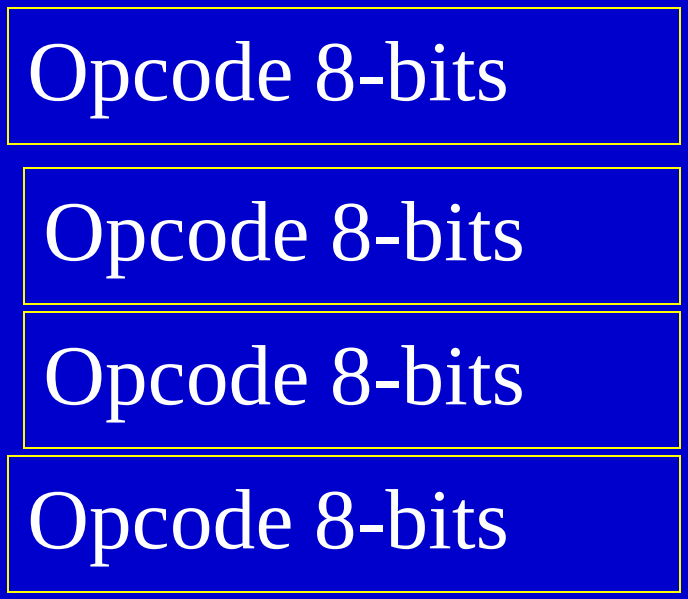
- We hand-simulate its execution
- Three PUSH statements
- Stack contains the values 3, 7, 2 (starting from the top of the stack)
- The MUL instruction pops the 3 and 7 off the stack
- Multiplies them
- Pushes the result (21) onto the stack

Verification

- Making the stack contents 21, 2.
- The ADD pops these two values off of the stack and adds them,
- Places the result (23) onto the stack, which is the result of the computation. This matches the result of the original expression, so the program is correct

Advantages 4 Instructions/word

- Stack Organized Processors Smallest Encoding Length



Push and Pop

- Push and Pop Instructions

Push or Pop Opcode 8-bits	Immediate Operands 24-bits

Summary

We learnt

- The operands and destination of an instruction in a stack-based organization implicit
- Instructions take fewer bits to encode than they do in general-purpose register organizations
- Stack organized processor instruction set
- Postfix notation for writing easily the instructions

End of Lesson 18 on

Stack-based processor Organisation