Chapter 4

The Processor

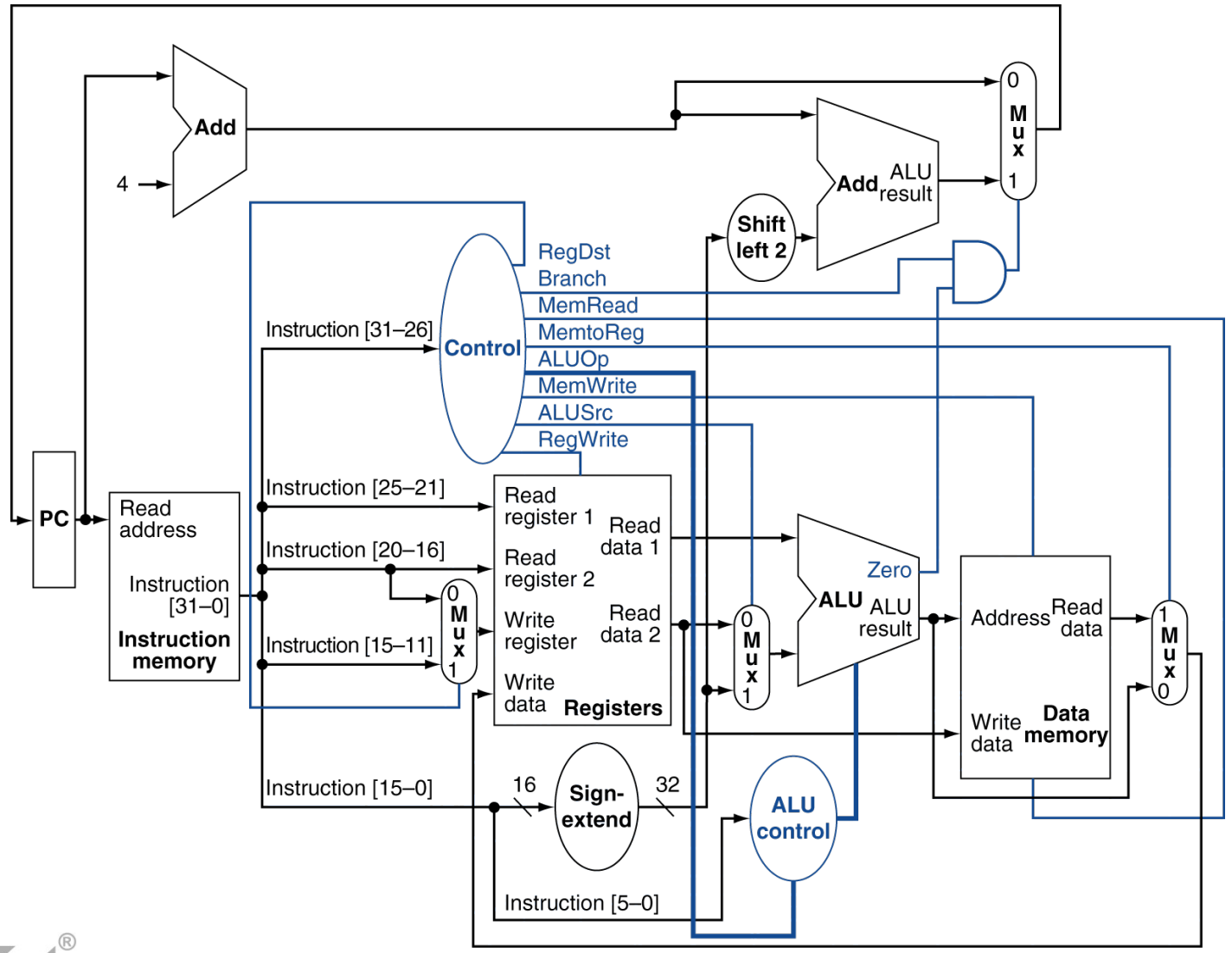
Introduction

- CPU performance factors
  - Instruction count
    - Determined by ISA and compiler
  - CPI and Cycle time
    - Determined by CPU hardware
- We will examine three MIPS implementations
  - A single cycle version
  - Multi-Cycle version
  - A more realistic pipelined version
- Simple subset, shows most aspects
  - Memory reference: `lw`, `sw`
  - Arithmetic/logical: `add`, `sub`, `and`, `or`, `slt`
  - Control transfer: `beq`, `j`

Instruction Execution

- PC → instruction memory, fetch instruction
- Register numbers → register file, read registers
- Depending on instruction class
  - Use ALU to calculate
    - Arithmetic result
    - Memory address for load/store
    - Branch target address
  - Access data memory for load/store
  - PC ← target address or PC + 4

Datapath With Control



Performance Issues

- The clock is determined by **the longest possible path** in the processor
  - Critical path: load instruction
  - Instruction memory → register file → ALU → data memory → register file
- Not feasible to vary period for different instructions
- Violates design principle
  - Clock cycle is the **worst case delay** for all instructions

Clock cycle time

- Assume time for stages is
  - 100ps for register read or write
  - 200ps for other stages

Instr	Instr fetch	Register read	ALU op	Memory access	Register write	Total time
lw	200ps	100 ps	200ps	200ps	100 ps	800ps
sw	200ps	100 ps	200ps	200ps		700ps
R-format	200ps	100 ps	200ps		100 ps	600ps
beq	200ps	100 ps	200ps			500ps

Single-Cycle Design: disadvantage

- Assuming fixed-period clock every instruction data-path uses one clock cycle implies:
  - CPI = 1
  - **cycle time determined by length of the longest instruction path (load)**
    - **but several instructions could run in a shorter clock cycle: *waste of time***
  - **resources used more than once in the same cycle need to be duplicated**
    - ***waste of hardware*** *and chip area*

Performance of Single-Cycle Machines

Assume that the operation times for the major functional units in this implementation are the following:

- Memory units: 200 picoseconds (ps)
- ALU and adders: 100 ps
- Register file (read or write): 50 ps

Assuming that the multiplexors, control unit, PC accesses, sign extension unit, and wires have no delay, which of the following implementations would be faster and by how much?

1. An implementation in which every instruction operates in 1 clock cycle of a fixed length.
2. An implementation where every instruction executes in 1 clock cycle using a variable-length clock, which for each instruction is only as long as it needs to be. (Such an approach is not terribly practical, but it will allow us to see what is being sacrificed when all the instructions must execute in a single clock of the same length.)

To compare the performance, assume the following instruction mix: 25% loads, 10% stores, 45% ALU instructions, 15% branches, and 5% jumps.

- CPU exec time = IC x CPI x Clock cycle time

Critical path for different instruction classes

Instruction class	Functional units used by the instruction class				
R-type	Instruction fetch	Register access	ALU	Register access	
Load word	Instruction fetch	Register access	ALU	Memory access	Register access
Store word	Instruction fetch	Register access	ALU	Memory access	
Branch	Instruction fetch	Register access	ALU		
Jump	Instruction fetch				

The required length for each instruction

Instruction class	Instruction memory	Register read	ALU operation	Data memory	Register write	Total
R-type	200	50	100	0	50	400 ps
Load word	200	50	100	200	50	600 ps
Store word	200	50	100	200		550 ps
Branch	200	50	100	0		350 ps
Jump	200					200 ps

- The clock cycle for a machine with a single clock for all instructions will be determined by the longest instruction, which is **600 ps**

- A machine with a variable clock will have a clock cycle that varies between 200 ps and 600 ps

- Thus, the average time per instruction with a variable clock is

CPU clock cycle = 600 × 25% + 550 × 10 % + 400 × 45 % + 350 × 15 % + 200 × 5 %

= 447.5 ps

- performance ratio:

$$\frac{\text{CPU performance}_{\text{variable clock}}}{\text{CPU performance}_{\text{single clock}}} = \frac{\text{CPU execution time}_{\text{single clock}}}{\text{CPU execution time}_{\text{variable clock}}}$$

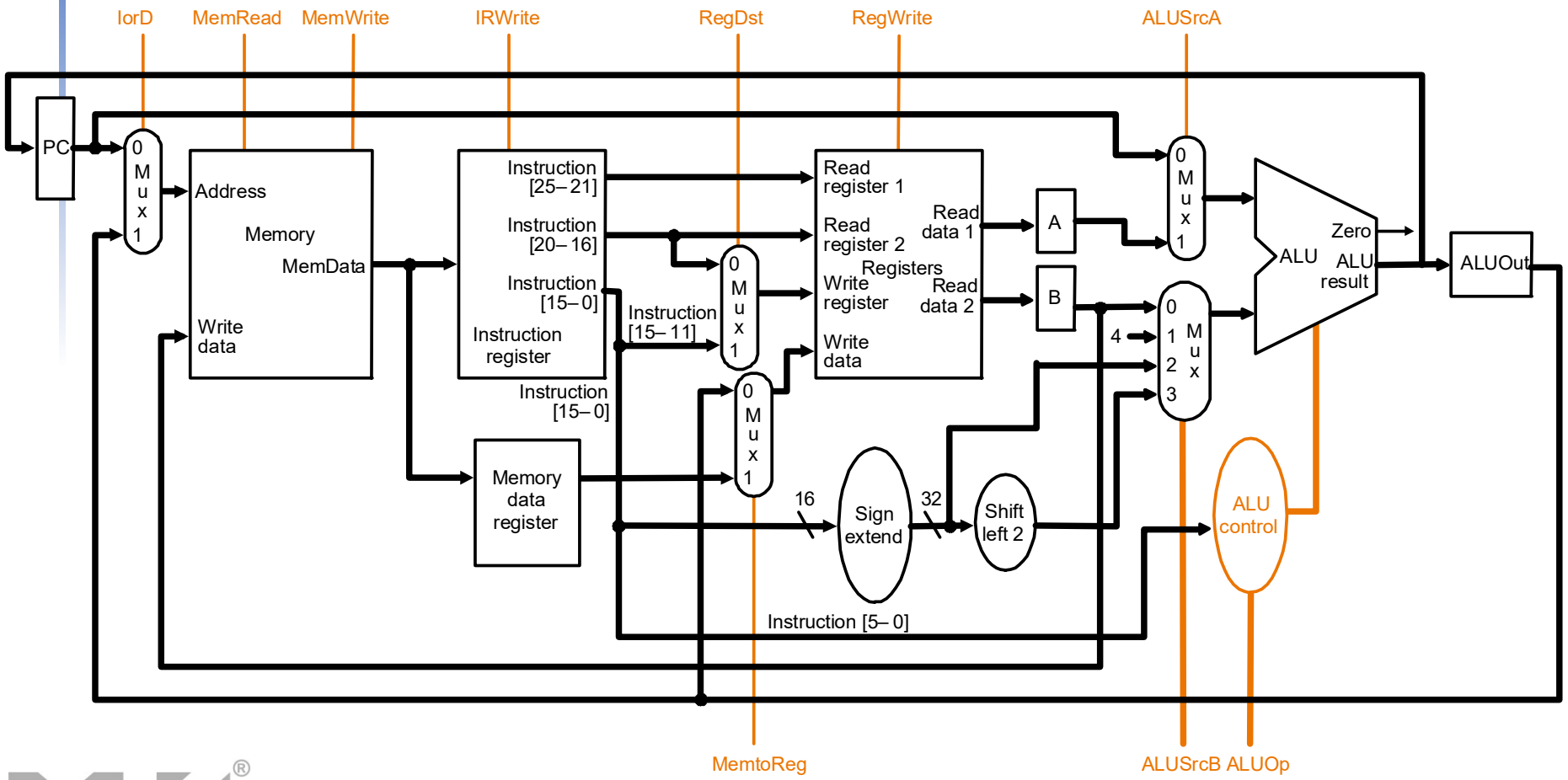
$$= \left(\frac{\text{IC} \times \text{CPU clock cycle}_{\text{single clock}}}{\text{IC} \times \text{CPU clock cycle}_{\text{variable clock}}} = \frac{\text{CPU clock cycle}_{\text{single clock}}}{\text{CPU clock cycle}_{\text{variable clock}}} \right)$$

= 600 / 447.5 = 1.34

Multi cycle implementation

- **An implementation in which an instruction is executed in multiple clock cycle**
- Break up the instructions into *steps*
  - each step takes one clock cycle
  - It allows a functional unit to be used more than once per instruction, since it is used on different clock
  - This Sharing can reduce amount of hardware required

Multicycle Datapath with Control



… with control lines and the ALU control block added – *not all* control lines are shown

- CPI in a Multicycle CPU Using the SPECINT2000 instruction mix shown below, what is the CPI, assuming that each state in the multicycle CPU requires 1 clock cycle?
- The mix is 25% loads (1% load byte + 24% load word), 10% stores (1% store byte + 9% store word), 11% branches (6% beq, 5% bne), 2% jumps (1% jal + 1% jr), and 52% ALU (all the rest of the mix, which we assume to be ALU instructions).

- The number of clock cycles for each instruction class
- Loads: 5
- Stores: 4
- ALU instructions: 4
- Branches: 3
- Jumps: 3

$$\text{CPI} = 0.25 \times 5 + 0.10 \times 4 + 0.52 \times 4 + 0.11 \times 3 + 0.02 \times 3 = 4.12$$

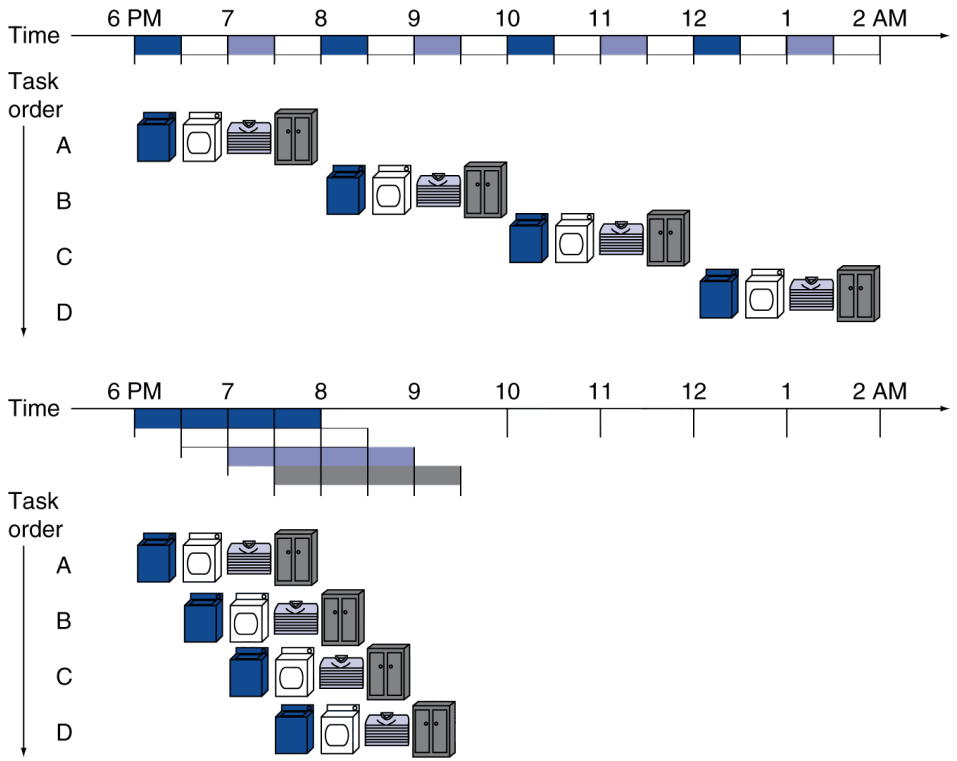
- This CPI is better than the worst-case CPI of 5.0 when all the instructions take the same number of clock cycles.

pipelining

- Pipelining is an implementation technique in which multiple instructions are overlapped in execution
- Today pipelining is key to making processors fast
- People who has done a lot of **laundry** has intuitively used pipelining

Pipelining Analogy

- Pipelined laundry: overlapping execution
  - Parallelism improves performance

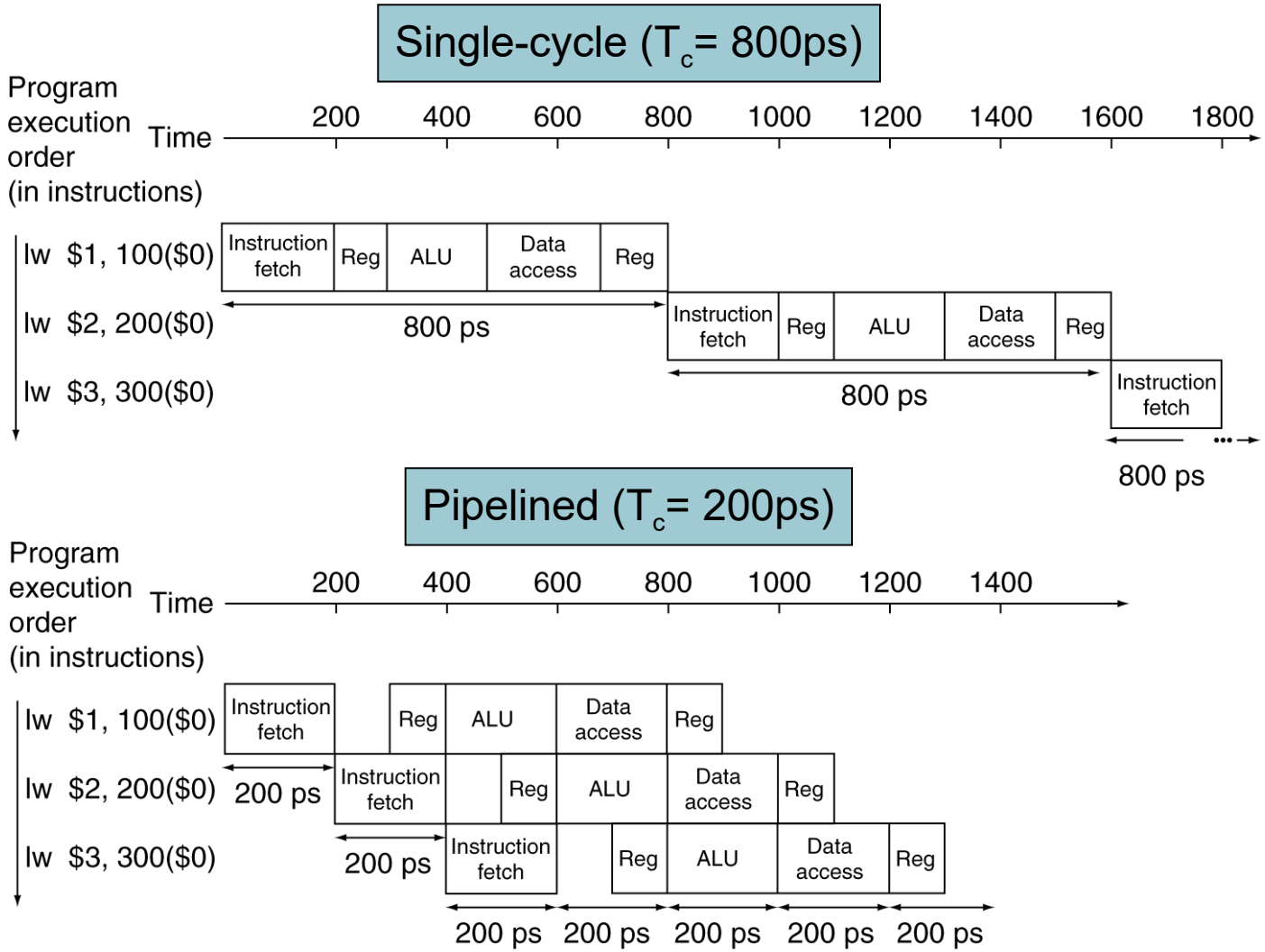


- Four loads:
  - Speedup
    = 8/3.5 = 2.3
- Non-stop:
  - Speedup
    = 2n/0.5n + 1.5 ≈ 4
    = number of stages

MIPS Pipeline

- Same principles apply to processors where we pipeline instruction execution
- MIPS instruction classically take 5 steps.
- Five stages, one step per stage
  1. IF: Instruction fetch from memory
  2. ID: Instruction decode & register read
  3. EX: Execute operation or calculate address
  4. MEM: Access memory operand
  5. WB: Write result back to register

Pipeline Performance



Pipeline Performance

- Assume time for stages is
  - 100ps for register read or write
  - 200ps for other stages
- Compare pipelined datapath with single-cycle datapath

Instr	Instr fetch	Register read	ALU op	Memory access	Register write	Total time
lw	200ps	100 ps	200ps	200ps	100 ps	800ps
sw	200ps	100 ps	200ps	200ps		700ps
R-format	200ps	100 ps	200ps		100 ps	600ps
beq	200ps	100 ps	200ps			500ps

Pipeline Speedup

- If all stages are balanced
  - i.e., all take the same time
  - Time between instructions$_{pipelined}$

    $= \frac{\text{Time between instructions}_{nonpipelined}}{\text{Number of stages}}$
  - *potential* speedup = number of pipe stages
- If not balanced, speedup is less
- Speedup due to increased throughput
  - Latency (time for each instruction) does not decrease

What happens when we increase no of instructions(1,000,000)