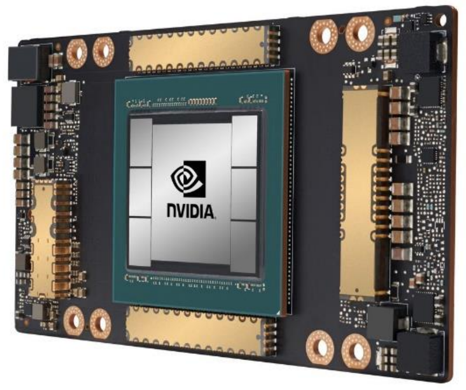
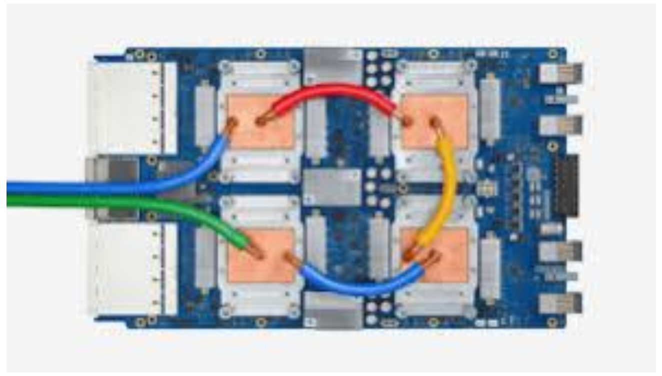
CSCB58: Computer Organization



Prof. Gennady Pekhimenko

University of Toronto

Fall 2020



The content of this lecture is adapted from the lectures of Larry Zheng and Steve Engels

CSCB58 Week 12

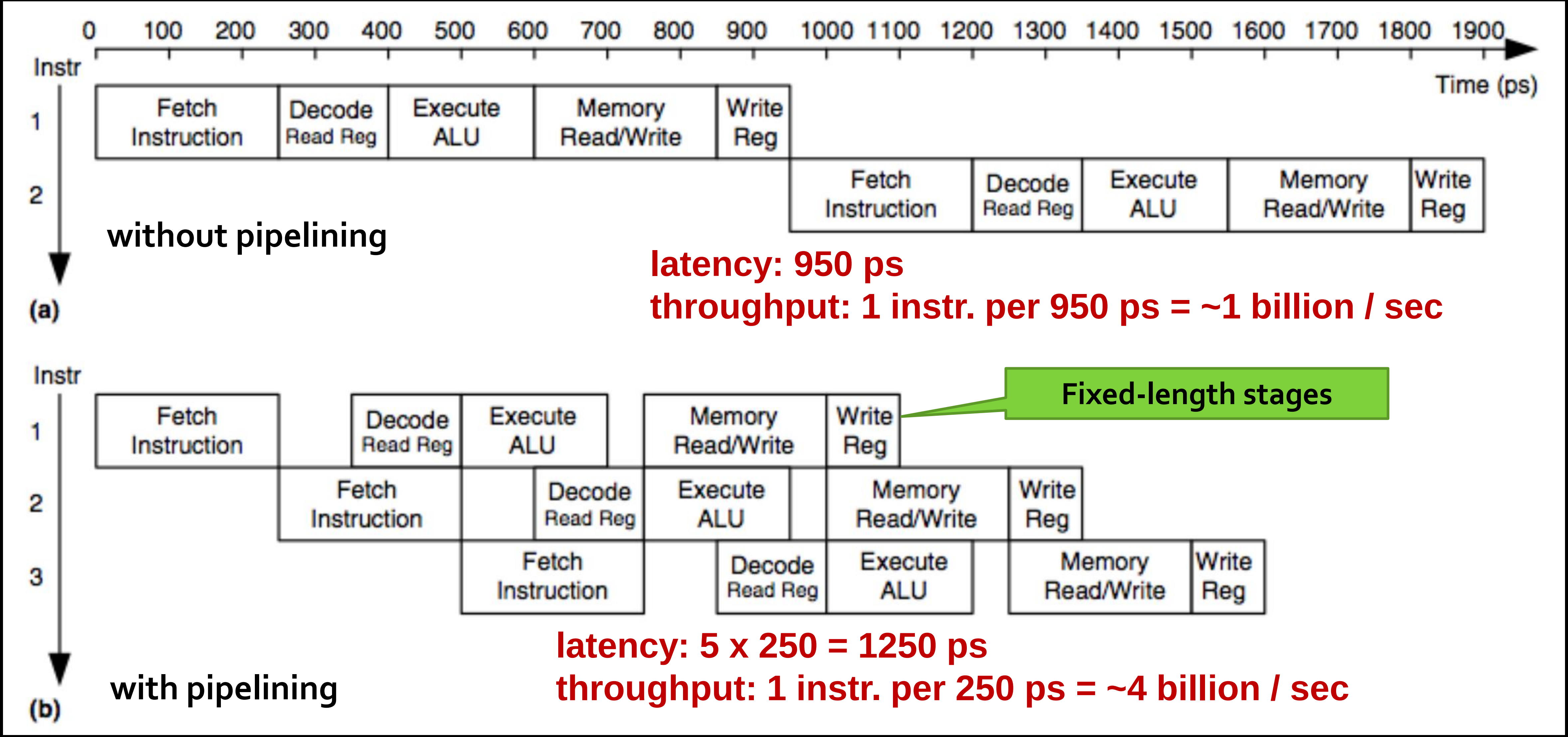
B58 Evaluation

- Please check your e-mail for a link to your evaluations
- Or go to: http://uoft.me/openevals

Execution Stages

- **Fetch**: Updating the PC and locating the instruction to execute.
- **Decode**: Translating the instruction and reading inputs from the register file.
- **Execute** / Address Computation: Using the ALU to compute an operation or calculate an address.
- **Memory Read or Write**: Memory operations must access memory. Non-memory operations skip this.
- **Register Writeback**: The result is written to the register file.

Pipelining the Execution Stages



without pipelining

latency: 950 ps
throughput: 1 instr. per 950 ps = ~1 billion / sec

Fixed-length stages

with pipelining

latency: 5 x 250 = 1250 ps
throughput: 1 instr. per 250 ps = ~4 billion / sec

fibonacci(int n)

```
int fib (int n){
        if (n <= 1)
                return n;
        else
        return fib (n - 1) + fib (n - 2);
}
```

fibonacci(int n)

```
fib:
           bgt $a0, 1, recurse
           move $v0, $a0
           jr $ra$t0, 0($sp)
```

1. Assign register names to variables and determine which is base case and which is recursive.
2. Only one input, n is passed in register $ao. The base case is the "then" clause. The recursive case is the "else" clause.
3. Convert the code for the base case.

Note: codes on the slides are not guaranteed to be correct. You need to be able to find the errors and fix them.

fibonacci(int n)

```
fib:
            bgt $a0, 1, recurse
            move $v0, $a0
            jr $ra$t0, 0($sp)
recurse:
            sub $sp, $sp, 12    # We need to store 3 registers
                                # to stack
            sw $ra, 0($sp)      # $ra is the first register
            sw $a0, 4($sp)      # $a0 is the second register,
                                # we cannot assume $a
                                # registers will not be
                                # overwritten by callee
```

Save callee- and caller-saved registers on the stack.

Note: codes on the slides are not guaranteed to be correct. You need to be able to find the errors and fix them.

fibonacci(int n)

```
fib: …


recurse: …
            addi $a0, $a0, -1 # N-1
            jal fib
            sw $v0, 8($sp)     # store $v0, the third register
                                # to be stored on the stack so
                               # it doesn't get overwritten by
                               # callee
```

Call fib recursively

Note: codes on the slides are not guaranteed to be correct. You need to be able to find the errors and fix them.

fibonacci(int n)

```
fib: …

recurse: …
         …
            lw $a0, 4($sp)     # retrieve original value of N
            addi $a0, $a0, -2 # N-2
            jal fib
```

Call fib recursively again

Note: codes on the slides are not guaranteed to be correct. You need to be able to find the errors and fix them.

fibonacci(int n)

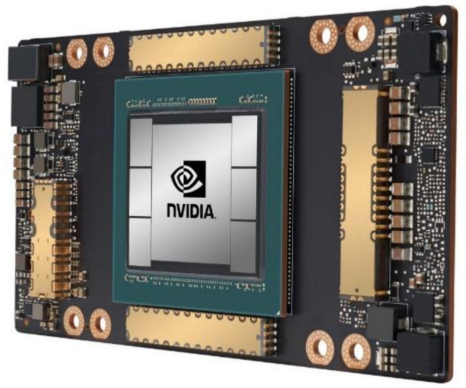
```
fib: …

recurse: …
         …
         …
         lw $t0, 8($sp) # retrieve first function result
         add $v0, $v0, $t0
         lw $ra, 0($sp) # retrieve return address
         addi $sp, $sp, 12
        jr $ra
```

Clean up the stack and return the result.

Note: codes on the slides are not guaranteed to be correct. You need to be able to find the errors and fix them.

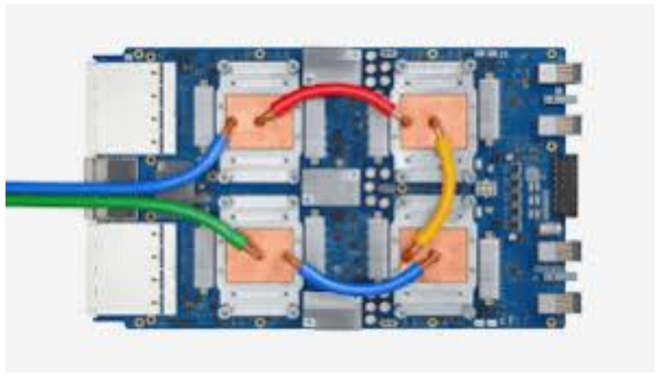
CSCB58: Computer Organization



Prof. Gennady Pekhimenko

University of Toronto

Fall 2020



The content of this lecture is adapted from the lectures of Larry Zheng and Steve Engels