# PIPELINED ARITHMETIC AND LOGIC UNIT


Yash 18110080 | Vrutik 18110191 | Viraj 18110188 | Vagisha 18110179 | Shruti Gupta 18110163


# CONCEPT

## Objective

Pipeline Arithmetic and Logic Unit to
- To improve the performance of CPU
- Optimal use of hardware
- Simultaneous execution of more than one instruction

## Pipelining

- Division of the whole process into smaller operations, each taking place simultaneously
- Does not decrease time of operation. Instead, it increases the throughput
- All modern processors extensively use pipelining
  Ex- Intel Core i7 (2nd generation) has 14 pipeline stages

**Timing Diagram for Instruction Pipeline Operation**

Time →

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Instruction 1 | FI | DI | CO | FO | EI | WO | | | | | | | | |
| Instruction 2 | | FI | DI | CO | FO | EI | WO | | | | | | | |
| Instruction 3 | | | FI | DI | CO | FO | EI | WO | | | | | | |
| Instruction 4 | | | | FI | DI | CO | FO | EI | WO | | | | | |
| Instruction 5 | | | | | FI | DI | CO | FO | EI | WO | | | | |
| Instruction 6 | | | | | | FI | DI | CO | FO | EI | WO | | | |
| Instruction 7 | | | | | | | FI | DI | CO | FO | EI | WO | | |
| Instruction 8 | | | | | | | | FI | DI | CO | FO | EI | WO | |
| Instruction 9 | | | | | | | | | FI | DI | CO | FO | EI | WO |

❑ Assume equal duration
❑ reduce execution time for 9 instructions: 54 time units to 14 time units.

## Methodology

- Hardware has divided into different stages to different stages perform different instructions at the same time
- Three-stage pipeline - IF/Decode, ALU, Write back stage
- Synchronized system - every task is completed on a clock tick
- Implementation of memory - managed by arrays
- Instruction and Data memory managed separately
- **ARM7TDMI** architecture followed

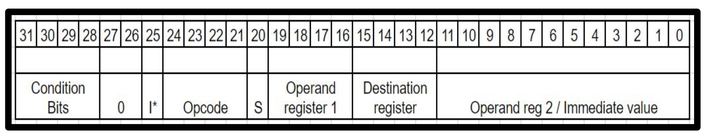
## Operations and Opcodes

| | |
|---|---|
| 1- AND (logic and) | 9 - TST(and w/o wb) |
| 2 - EOR (logic xor) | 10 - TEQ (xor w/o wb) |
| 3 - SUB (subtract) | 11 - CMP (sub w/o wb) |
| 4 - RSB (rev. sub) | 12 - CMN (add w/o wb) |
| 5 - ADD (add) | 13- ORR (OR) |
| 6 - ADC (add + carry) | 14 - MOV (operand 2) |
| 7 - SBC (sub with carry) | 15 - BIC (bit clear) |
| 8 - RSC (rev sub with carry | 16 - MVN(~Operand 2) |
| | *wb - write back |

# CODE

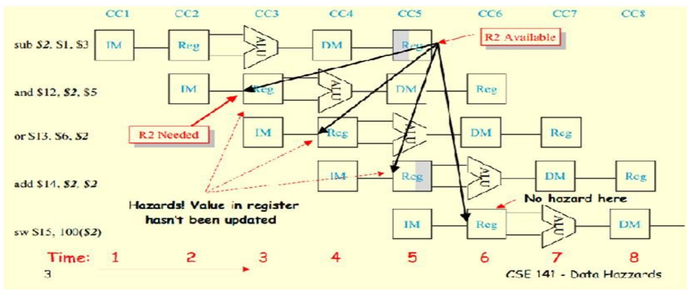
## Stage 1
### Instruction Fetch and Decode

- Task: Fetches the instruction from memory
- Control through Program Counter
- ARMv7 instruction format has been used



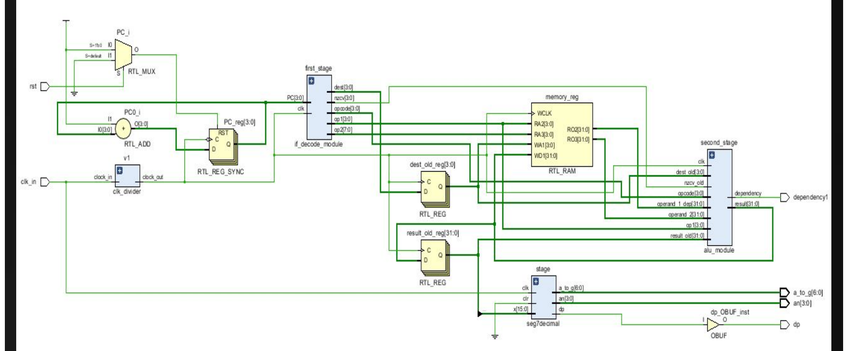

## Stage 2
### Execute

- Implements 16 standard Data Processing Operations (defined in ARM 7)
- Updates the values of flags **nzcv** and **is_write**
- Checks for inter-dependent instructions
  - The operand value is read from the same register where the last instruction is storing the data
  - Solution - Direct value access for the next instruction
  - Conditional assignment of the operand value
  - Prevents bubbles in the process
  - Speeds up the process




## Stage 3
### Write Back

- This is responsible for the memory write operation
- After the completion of the calculation stage - ALU execution
- To ensure there is no ambiguity (As update and read instructions are executed simultaneously)
  - Read at the negative edge of the clock
  - Write at the positive edge of the clock




Complete Pipeline RTL diagram

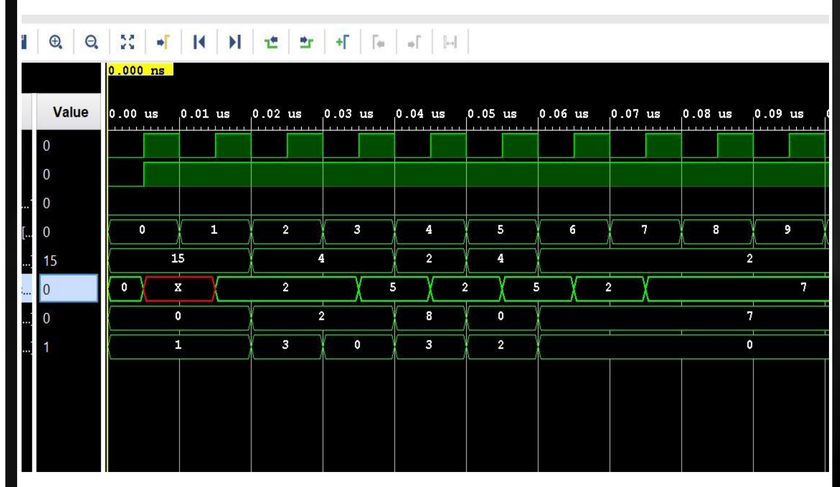
## Pipelining Module

- The control module, which calls other modules
- Synchronises all other modules w.r.t. the clock
- Controls the Program Counter - increments as soon as the current instruction is executed
- Contains memory array - Memory read is performed from this module, without the clock tick
- Extra FFs - for result data and destination reg
  - result_old and dest_old
  - to prevent overwriting of variables

# CONVERT

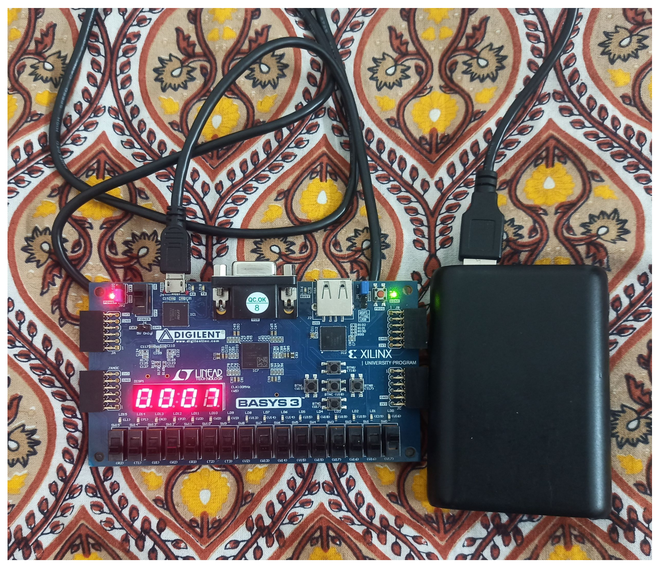
## Simulation Output and Graphs

- The first instruction is decoded in the first clock cycle
- The ALU executes the first instruction in the second cycle
- Meanwhile, the Instruction fetch stage works on the second instruction
- The pipelined ALU gives the first output in the third cycle
- Then, the system gives the output of consecutive instructions in consecutive clocks.




## Implementation on Basys3 Board

- The 32-bit instructions are stored in an array and stored on the FPGA.
- The pipelined ALU executes the instructions and shows the result on the FPGA board
- The 7-segment displays show the 16 least significant bits of the result
- The LED shows the dependency flag (5 - No dependency detected).




## References


- www.github.com/chsasank/ARM7
- https://github.com/kejriwalrahul/3-Stage-Pipeline
- http://vision.gel.ulaval.ca/~jflalonde/cours/1001/h17/docs/arm-instructionset.pdf
- https://www.scss.tcd.ie/~waldroj/3d1/arm_arm.pdf
- https://www.pantechsolutions.net/getting-started-with-arm-architecture
- https://github.com/mhyousefi/MIPS-pipeline-processor
- http://www.cs.cornell.edu/courses/cs3410/2012sp/project/pa1.html