Embedded Systems

Ch 15
ARM Organization and Implementation

Byung Kook Kim
Dept of EECS
Korea Advanced Institute of Science and Technology

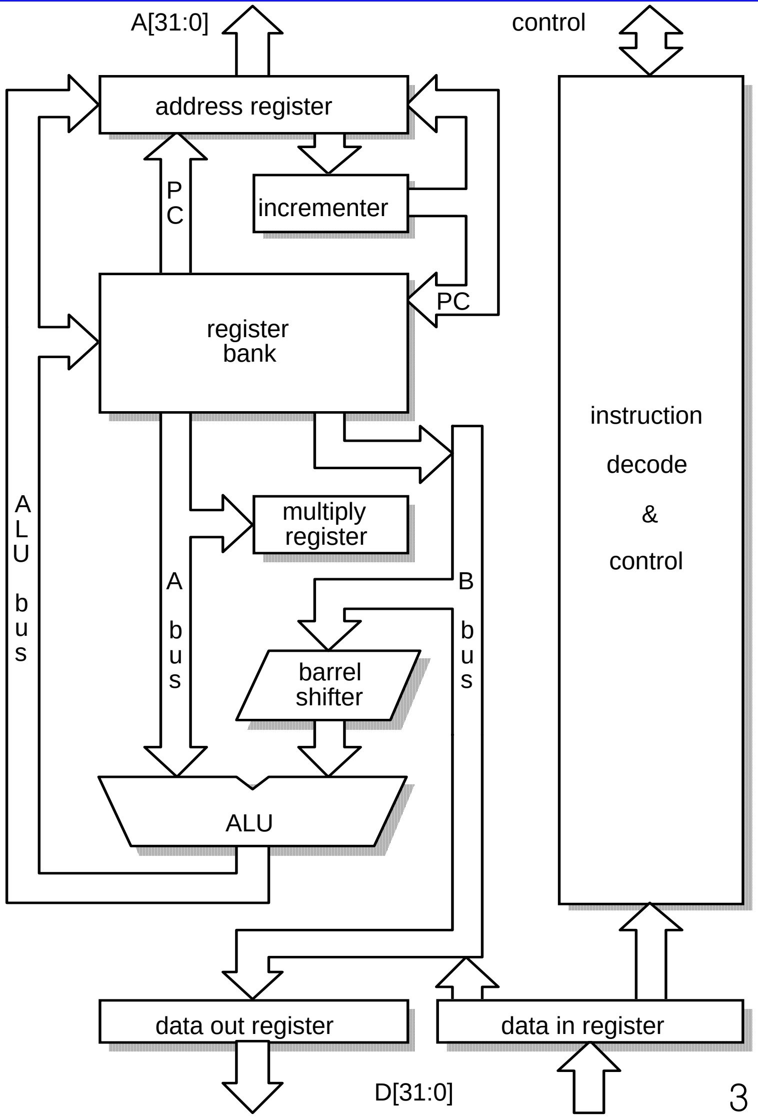
Summary

- **ARM architecture**
  - Very little change
    - From the first 3-micron devices t Acorn Computers, 1983-85
    - To the ARM6 & ARM7 by ARM Limited, 1990-95
  - 5 stage pipeline
  - CMOS technology reduced size by ~1/10
  - Performance of cores improved dramatically
  - 1995- : Separate instruction and data memories

- *In this chapter*
  - Describes internal architectures
  - Covers the general principles of operation of 3-state and 5-stage pipelines

1. 3-Stage Pipeline ARM Organization

- **ARM with 3-stage pipeline**
  - The register bank
    - Two read ports: sources
    - One write port: destination
    - PC (r15) has an additional read port, an additional write port: instruction fetch and fetch address increment.
  - The barrel shifter
    - Shift/rotate by any number of bits
  - ALU
    - Arithmetic/logic operations
  - Address register and incrementer
    - Select and hold memory address.
    - Sequential addressing
  - Data register
    - Data from/to memory
  - Instruction decoder & control logic



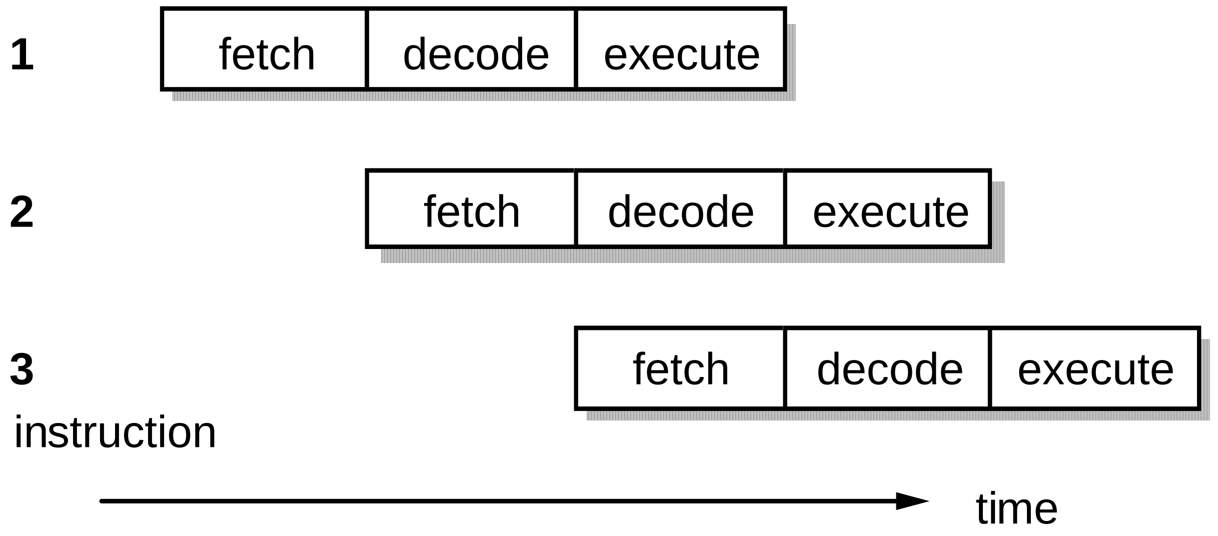
3-Stage Pipeline ARM Organization (II)

- **The 3-stage pipeline**
  - ARM processors up to the ARM7
  - Pipeline stages
    - 1. **Fetch**: The instruction is fetched from memory and placed in the instruction pipeline.
    - 2. **Decode**: The instruction is decoded and the datapath control signals prepared for the next cycle. In this stage, the instruction ‘owns’ the decode logic but not the datapath.
    - 3. **Execute**: The instruction ‘owns’ the datapath; the register bank is read, an operand shifted, the ALU result generated and written back into a destination register.
  - At any one time, three different instructions may occupy each of these stages: The hardware in each stage has to be capable of independent operation.

3-Stage Pipeline ARM Organization (III)

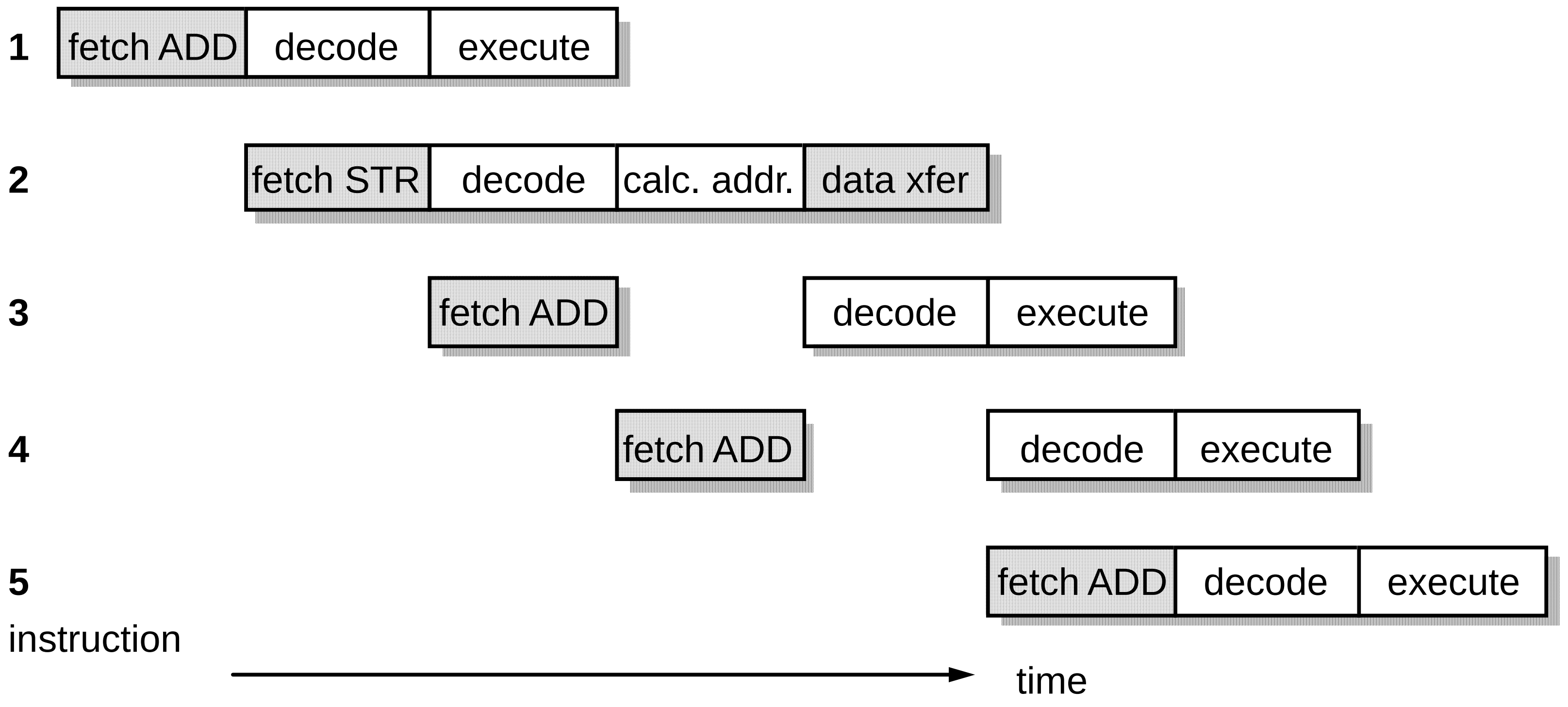
- **The 3-stage pipeline (II)**
  - **Latency**: 3-cycles to complete one instruction
  - **Throughput**: one instruction per cycle

  - Single-cycle instruction 3-stage pipeline operation



3-Stage Pipeline ARM Organization (IV)

- **The 3-stage pipeline (III)**
  - Multi-cycle instruction: ADD then STR



3-Stage Pipeline ARM Organization (V)

- **The 3-stage pipeline (IV)**
  - Breaks in the ARM pipeline
    - All instructions occupy the datapath for one or more adjacent cycles
    - For each cycle that an instruction occupies the datapath, it occupies the decode logic in the immediately preceding cycle
    - During the first datapath cycle, each instruction issues a fetch for the next instruction
    - Branch instructions flush and refill the instruction pipeline.
- **PC (Program Counter) behavior**
  - PC must run ahead of the current instruction: 8-bytes ahead.
  - For most normal purposes the assembler or compiler handles all details.

2. 5-Stage Pipeline ARM Organization

- **Demand for higher performance**
  - 3-stage pipeline: cost-effective
  - Time required to execute a given program:

$$T_{prog} = \frac{N_{inst} \times CPI}{f_{clk}}$$

    - CPI: Clock per instruction

  - Two ways to increase performance
    - Increase the clock rate
      - Logic in each pipeline to be simplified
    - Reduce the average number of clock cycles per instruction, CPI
      - Instructions which occupy more than one slot: To occupy fewer slots
      - Pipeline stalls caused by dependencies between instructions are reduced.

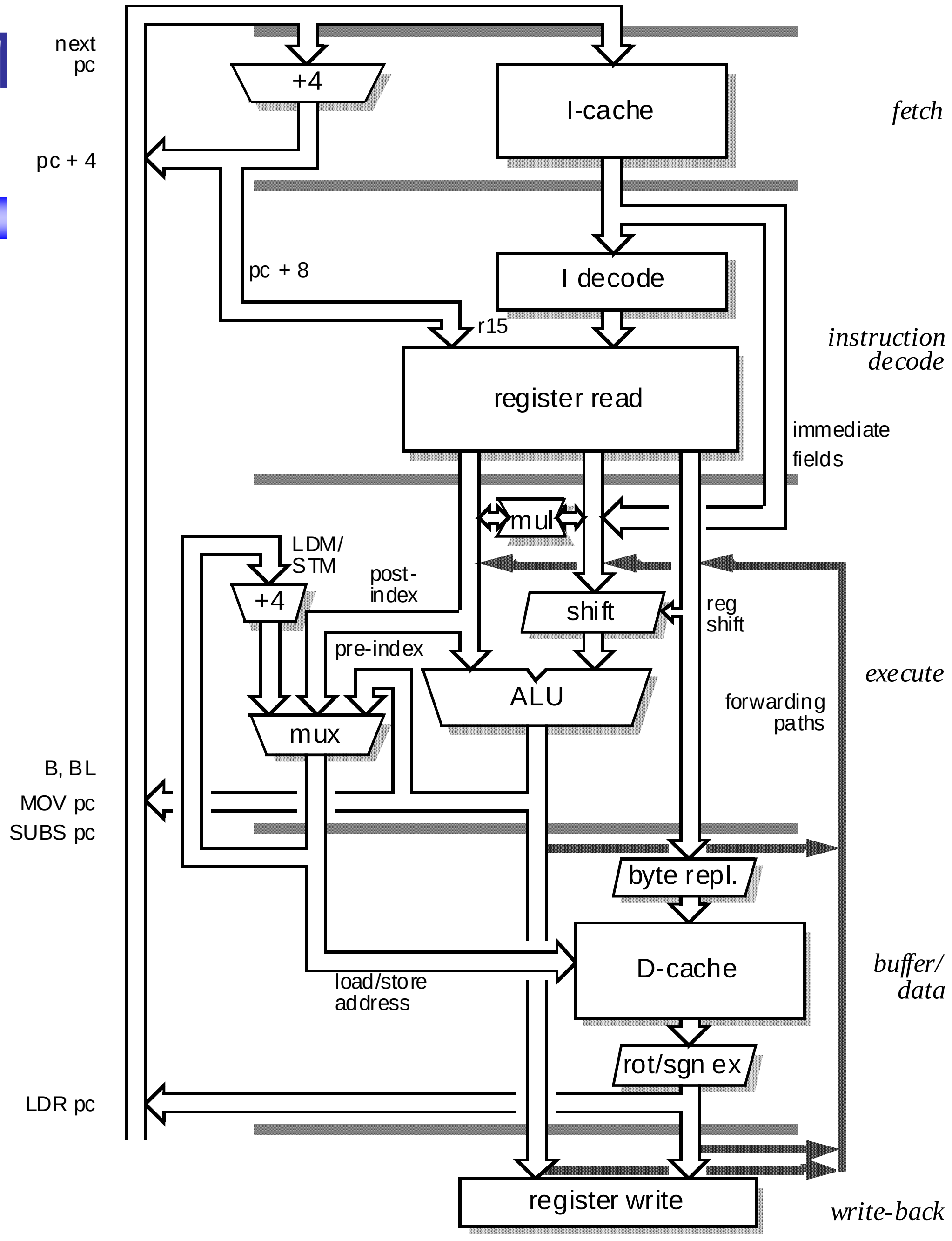
5-Stage Pipeline ARM Organization (II)

- **Memory bottleneck**
  - Von Neumann bottleneck
    - Any stored-program computer with a single instruction/data memory will have its performance limited by the available memory bandwidth
    - Fetch an instruction or to transfer data
  - Solution
    - Faster memory: more than one value in each clock cycle
    - Separate memories for instruction and data accesses
  - Higher performance ARM
    - 5-stage pipeline
      - Reduce max work in each stage
      - Higher clock frequency
    - Separate instruction and data memories
      - Reduced CPI

5-Stage Pipeline ARM Organization (III)

- **The 5-stage pipeline**
  - 1. Fetch
  - 2. Decode
    - 3 operand read ports
  - 3. Execute
    - Shift & ALU
    - Mem adr for load/store
  - 4. Buffer/data
    - Data memory access
    - Buffer for ALU result
  - 5. Write-back
    - Write-back to register or memory

  - Used for many RISC

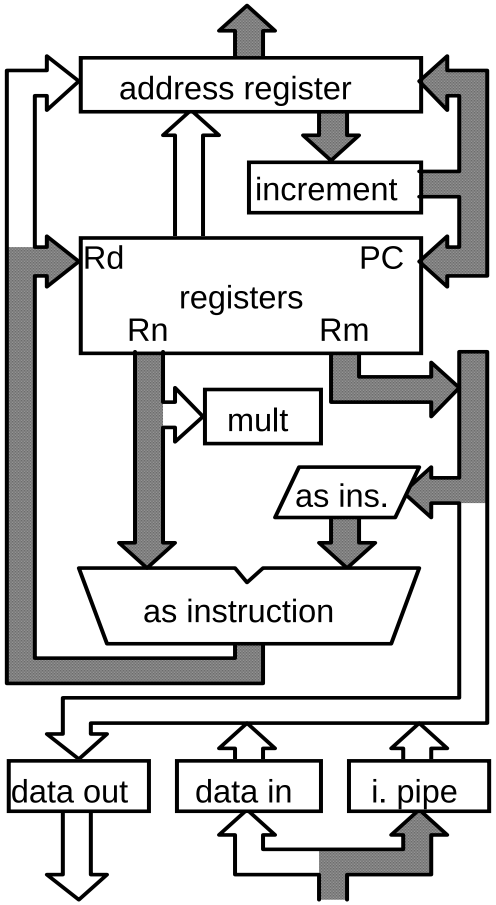


5-Stage Pipeline ARM Organization (IV)

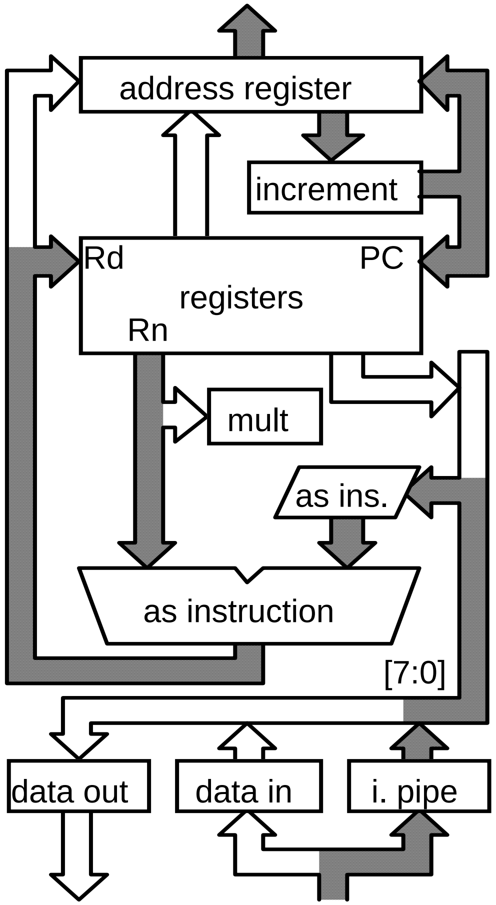
- **Data forwarding**
  - Instruction execution is spread across three pipeline stages
    - Resolve data dependencies: **forwarding paths**
  - When an instruction needs to use the result of one of predecessors before the result has returned to the register file: pipeline hazards
    - Forwarding paths allow results to be passed between stages as soon as they are available
  - Exception
    - Even with forwarding, it is not possible to avoid pipeline stall
    - `LDR    rN, [...]        ; Load rN from somewhere`
    - `ADD    r2, r1, rN       ; and use it immediately`
    - One cycle stall required
    - Encourage compiler (or assembly programmer) not to put a dependent instruction immediately after a load instruction

3. ARM Instruction Execution

- **Data processing instructions**



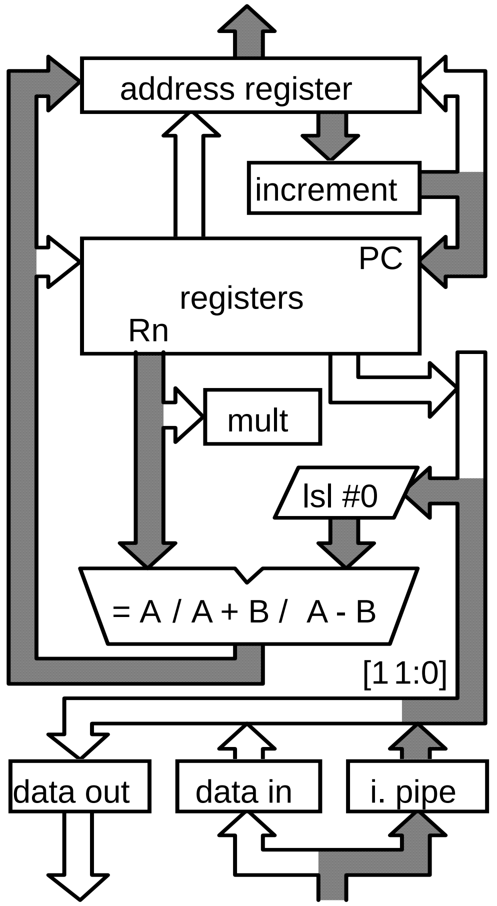
(a) r egister - r egister operations



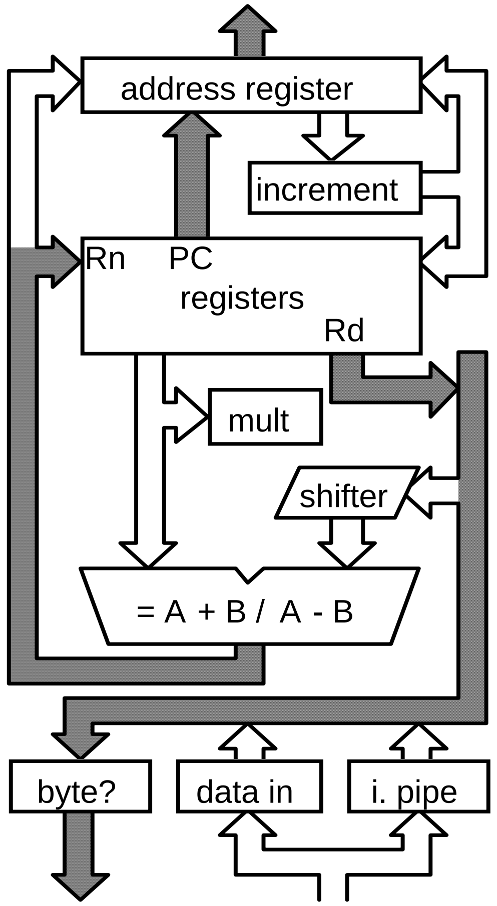
(b) r egister - immediate operations

ARM Instruction Execution (II)

- **Data transfer instructions (Store)**



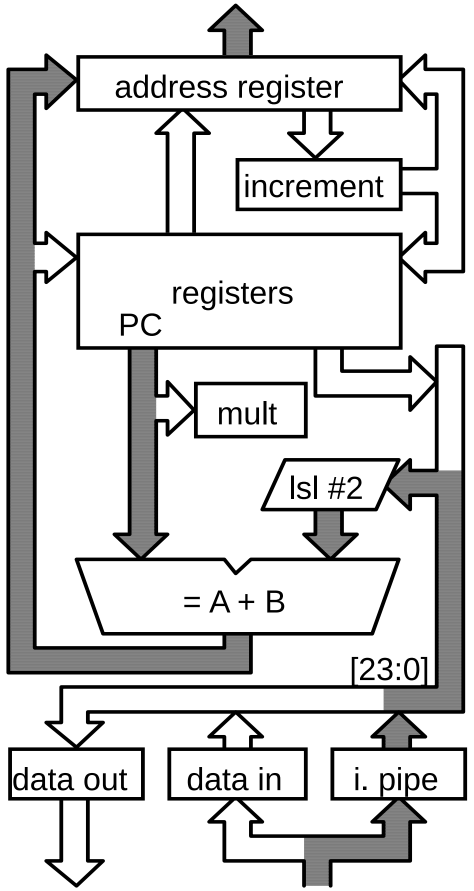
(a) 1st cycle - compute addr ess



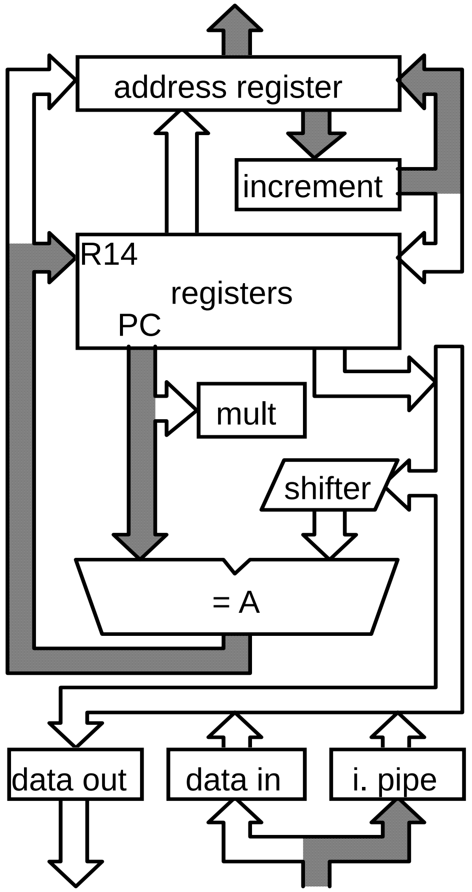
(b) 2nd cycle - stor e data & auto-index

ARM Instruction Execution (III)

- **Branch instructions (First two cycles)**



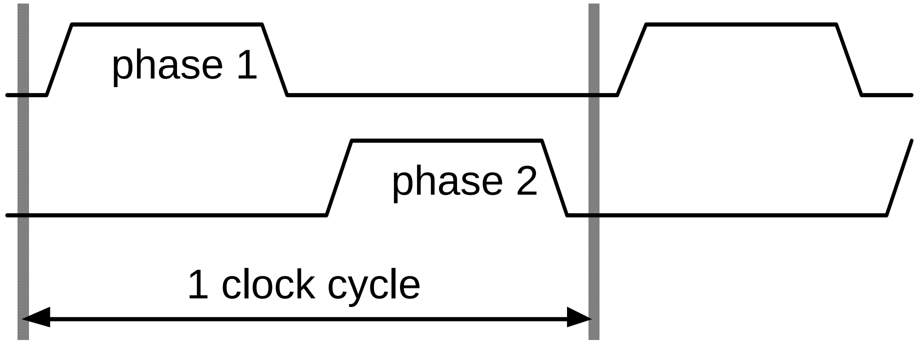
(a) 1st cycle - compute branch tar get



(b) 2nd cycle - save r eturn address

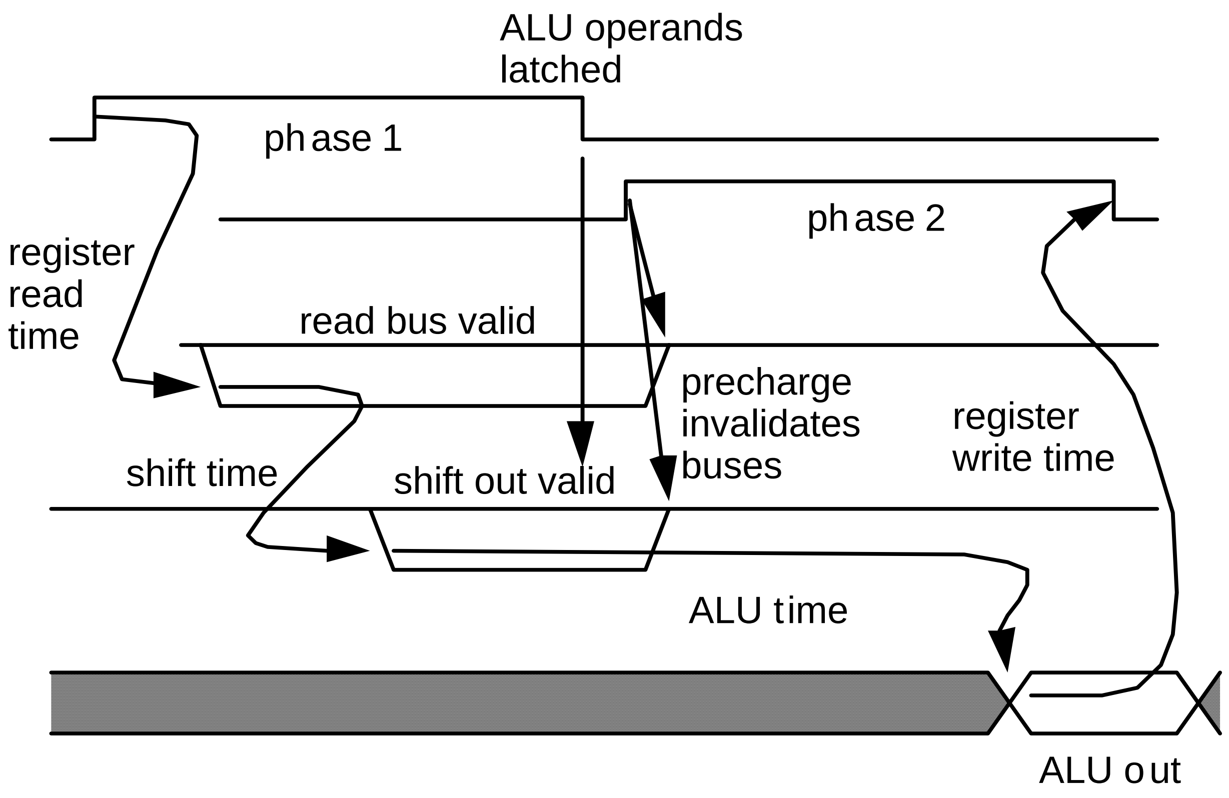
4. ARM Implementation

- **Design**
  - *Register Transfer Level* (RTL): Describe datapath section
  - *Finite State Machine* (FSM): Describe control section
- **Clocking scheme**
  - 2-phase non-overlapping clocks
    - Allows use of level-sensitive transparent latches
    - Data movement is controlled by passing data alternatively through latches which are open during phase 1 and then during phase 2
    - Non-overlapping: no race conditions in the circuit



ARM Implementation (II)

- **Datapath timing**
  - 3-stage pipeline datapath timing

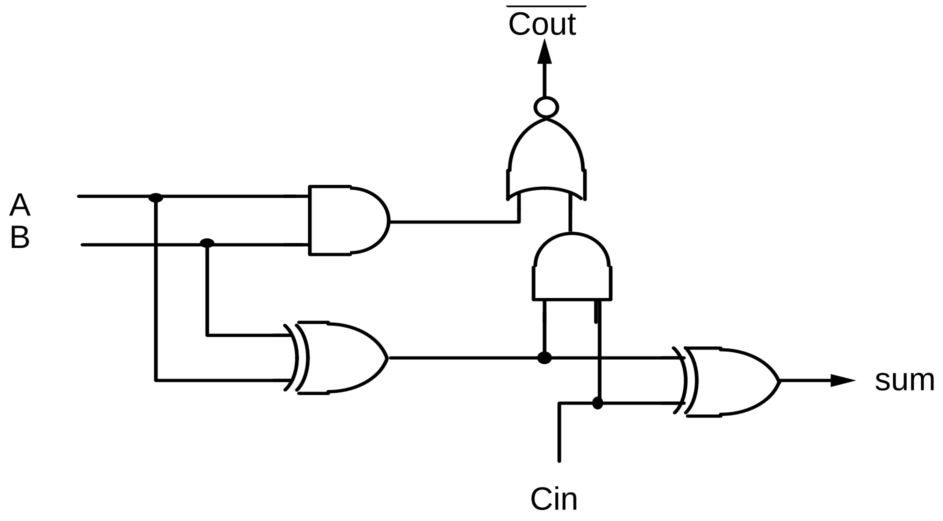


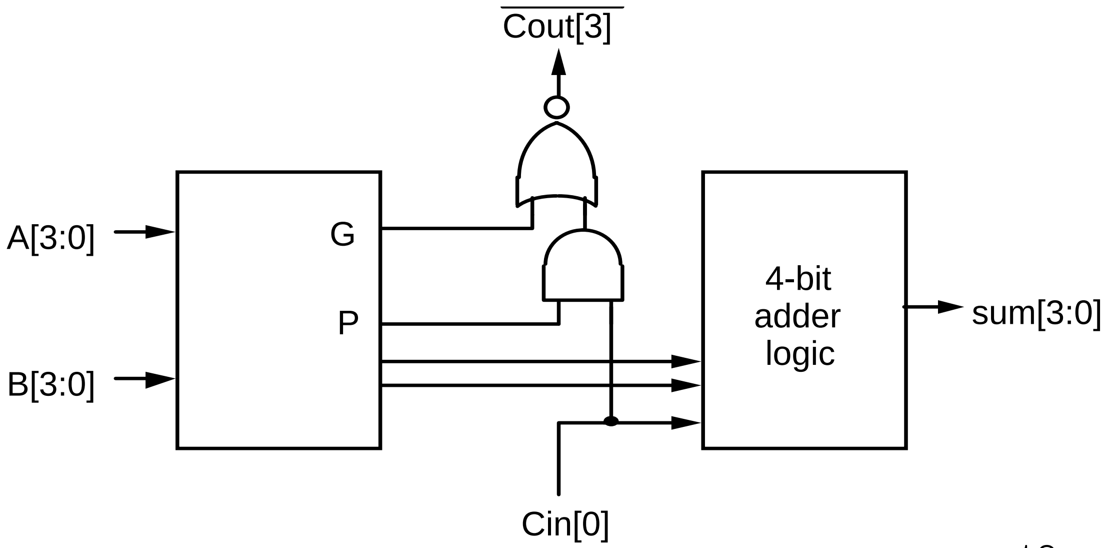
ARM Implementation (III)

- **Datapath timing (cont'd)**
  - The minimum datapath cycle time is the sum of:
    - The register read time
    - The shifter delay
    - The ALU delay
    - The register write set-up time
    - The phase 2 to phase 1 non-overlapping time.
  - ALU delay
    - Dominant
    - Highly variable
      - Logic operation: relatively fast
      - Arithmetic operation: carry propagation. Involve longer paths.

ARM Implementation (IV)

- **Adder design**
  - Ripple-carry adder (First ARM processor prototype)
    - CMOS AND-OR-INVERT gates
    - Worst-case carry path: 32 gates long
  - 4-bit carry look-ahead (ARM2)
    - G: carry generate
    - P: carry propagate
    - Worst-case carry path: 8 gate delays
    - AND-OR_INVERT gates & AND/OR logic

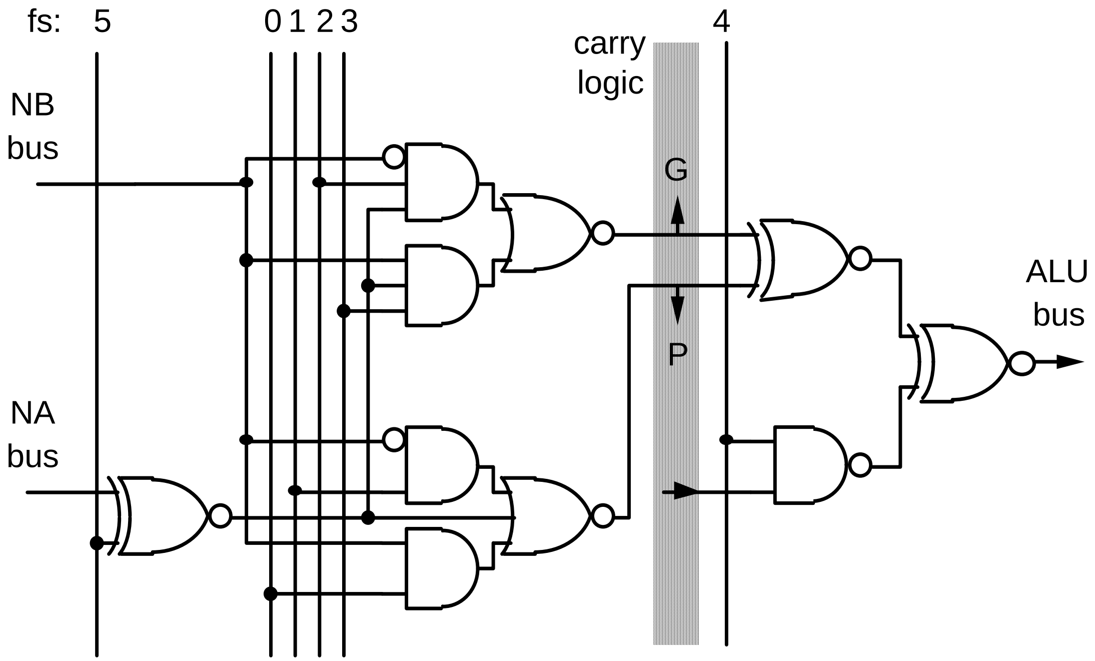




ARM Implementation (V)

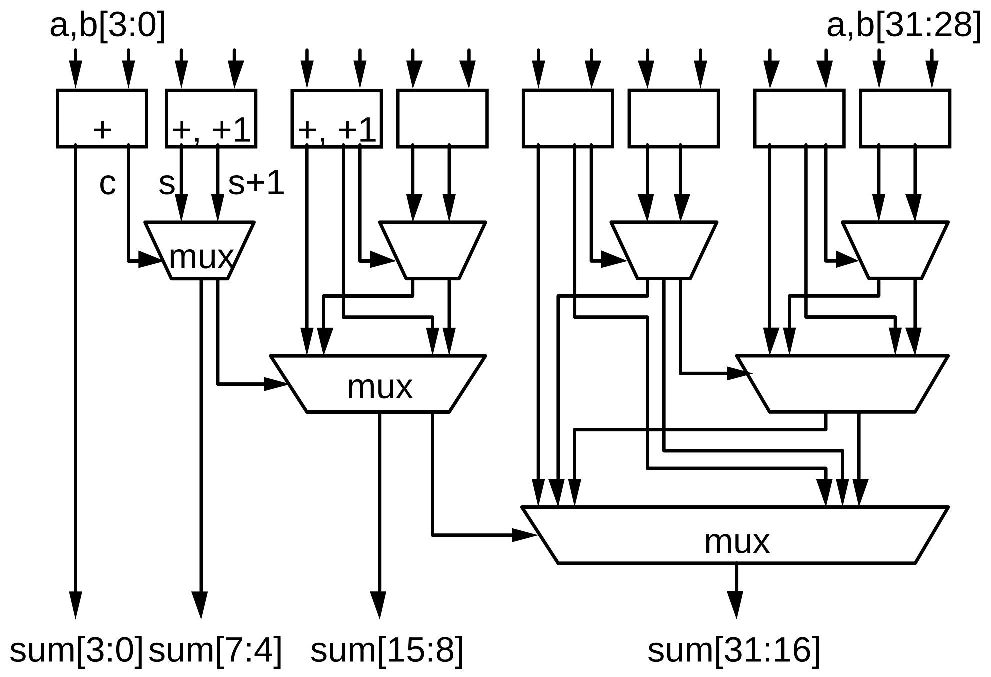
- **ALU functions (ARM2)**

fs5	fs4	fs3	fs2	fs1	fs0	ALU output
0	0	0	1	0	0	A and B
0	0	1	0	0	0	A and not B
0	0	1	0	0	1	A xor B
0	1	1	0	0	1	A plus not B plus carry
0	1	0	1	1	0	A plus B plus carry
1	1	0	1	1	0	not A plus B plus carry
0	0	0	0	0	0	A
0	0	0	0	0	1	A or B
0	0	0	1	0	1	B
0	0	1	0	1	0	not B
0	0	1	1	0	0	zero



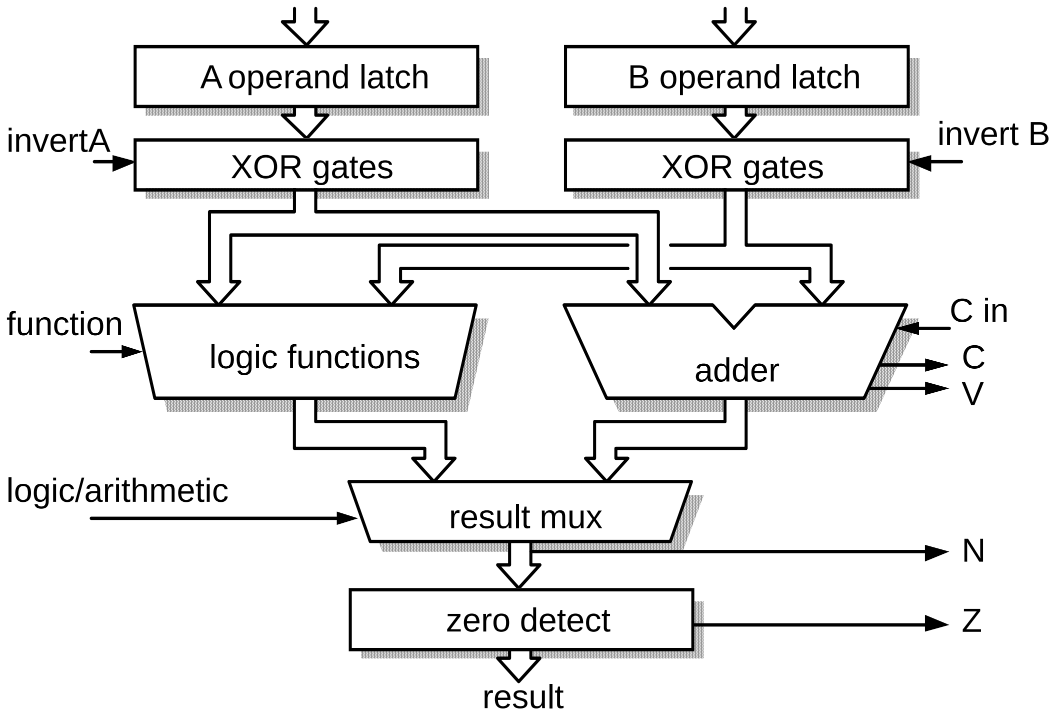
ARM Implementation (VI)

- **ARM6 carry-select adder**
  - Computes the sums of various fields of the word for a carry-in of both and one
  - The final result is selected by using the carry-in value to control a multiplexer
  - Critical path O($\log_2$[word width])



ARM Implementation (VII)

- **ARM6 ALU structure**
  - Carry-select adder does not easily lead to a merging of the arithmetic and logic functions into a single structure
  - Separate logic unit runs in parallel with the adder
  - Multiplexer selects the output.



ARM Implementation (VIII)

- **Carry arbitration adder (ARM9TDMI)**
  - Computes all intermediate carry values using a ‘parallel-prefix’ tree, which is a very fast parallel logic structure
  - Recodes the conventional propagate-generate information in terms of two new variables, u and v.

A	B	C	u	v
0	0	0	0	0
0	1	unknown	1	0
1	0	unknown	1	0
1	1	1	1	1

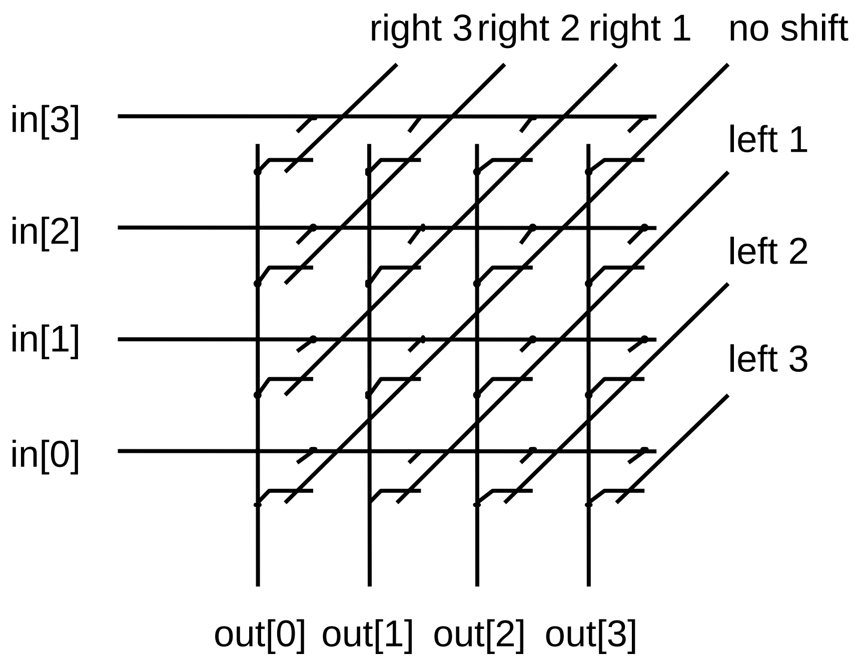
  - Combined with that from a neighboring bit position:

$$(u,v).(u',v') = (v+u.u', v+u.v') \qquad \text{(eq.12)}$$

  - This combinational operator is associative
  - u and v can be computed for all the bits in the sum using a regular parallel prefix tree.
  - u and v can be used to generate the (Sum, Sum+1) values required for a hybrid carry arbitration/carry select adder.

ARM Implementation (IX)

- **The barrel shifter**
  - Shift time contributes directly to the datapath cycle time
  - Cross-bar switch matrix is used to steer each input to the appropriate output
  - 4x4 switch ->
  - 32x32 switch for ARM.



ARM Implementation (X)

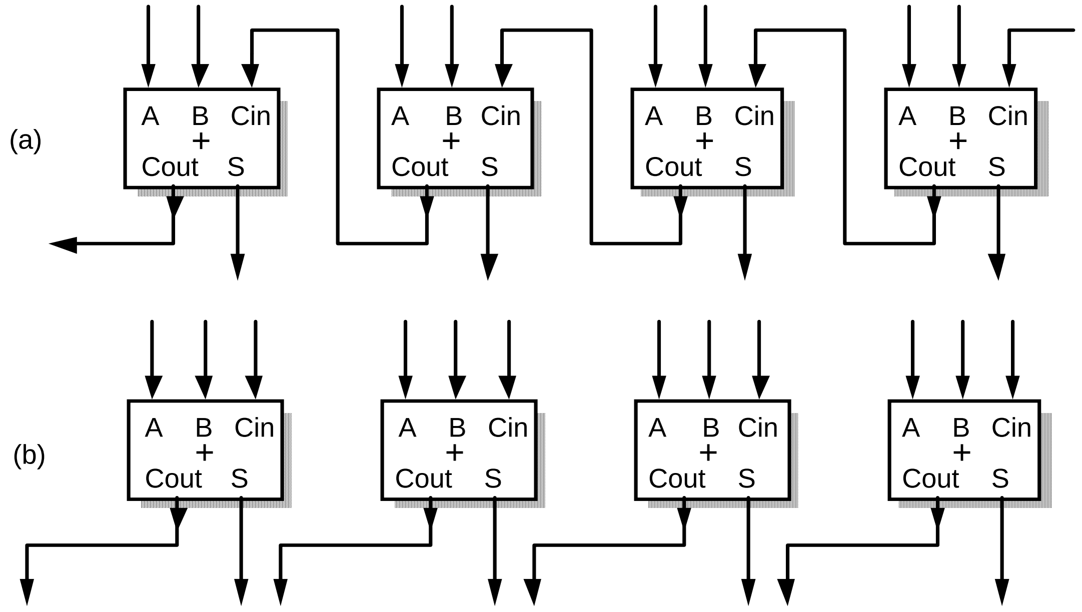
- **Multiplier design**
  - Older ARM cores include low-cost multiplication hardware that supports only the 32-bit result multiply and multiply-accumulate instructions
    - Uses the main datapath iteratively – shift & ALU
    - Modified Booth's algorithm to produce the 2-bit product
    - Overhead of few % area of the ARM core

Carry-in	Multiplier	Shift	ALU	Carry-out
0	x0	LSL#2N	A+0	0
	x1	LSL#2N	A+B	0
	x2	LSL#(2N+1)	A–B	1
	x3	LSL#2N	A–B	1
1	x0	LSL#2N	A+B	0
	x1	LSL#(2N+1)	A+B	0
	x2	LSL#2N	A–B	1
	x3	LSL#2N	A+0	1

  - Recent ARM cores have high-performance multiplication hardware and support the 64-bit result multiply and multiply-accumulate instructions

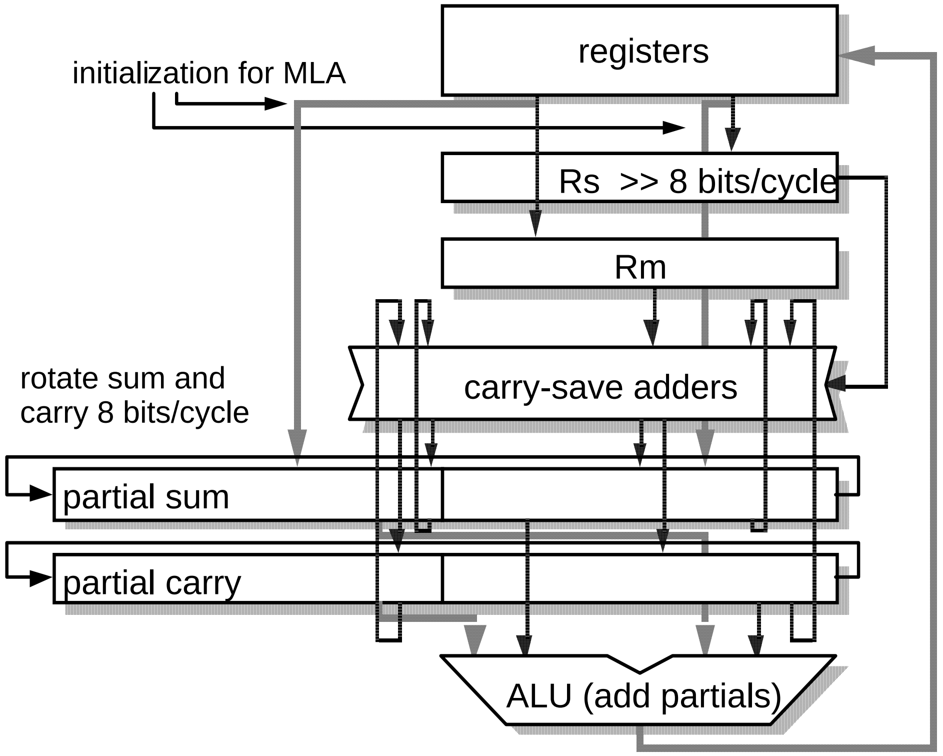
ARM Implementation (XI)

- **High-speed multiplier**
  - Carry-save adder
    - Carries only propagate across one bit per addition stage
    - Much shorted logic path than the carry-propagate adder
    - Can be performed in a single cycle
    - Produces a sum in redundant binary representation



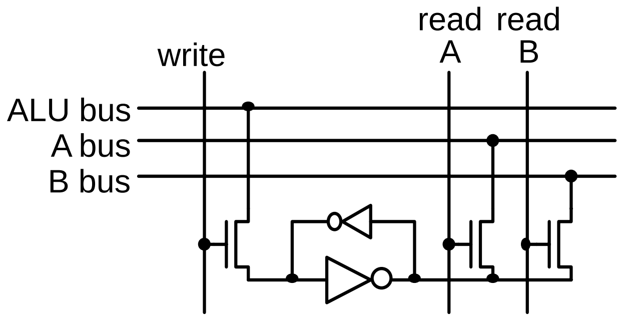
ARM Implementation (XII)

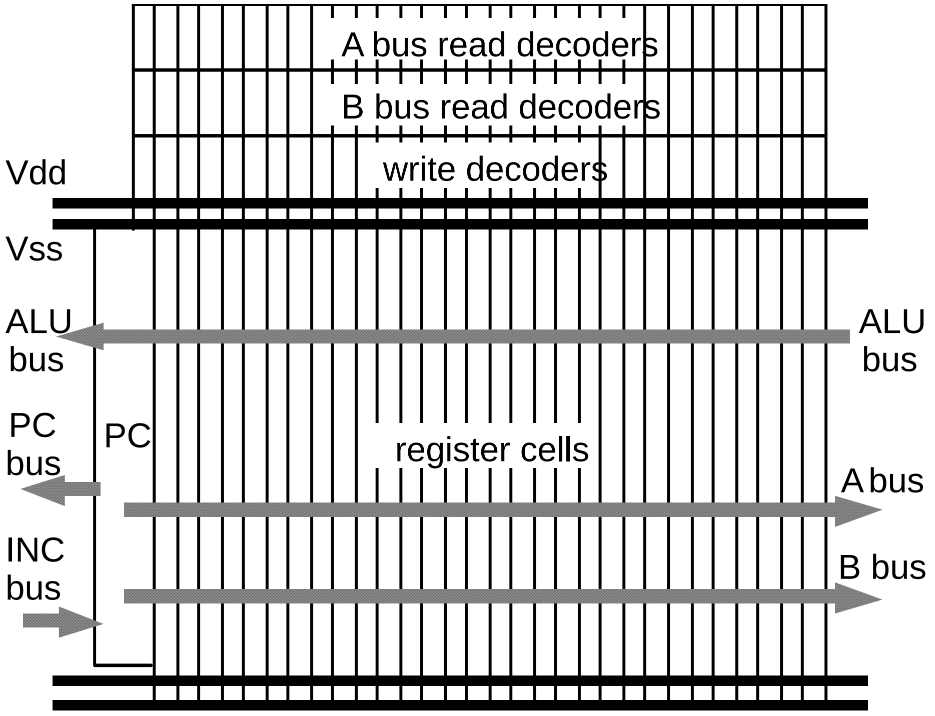
- **High-speed multiplier (cont'd)**
  - Several layers of carry-save adder in series, each handling one partial product
  - 4 layers of adders
  - Can multiply 8 bits/clock
  - More dedicated hardware
    - 160 bits of shift register
    - 128 bits carry-save adder logic
    - 10% of the simpler processor core
  - Speed up multiplication by a factor of ~3
  - Added functionality of the 64-bit result.



ARM Implementation (XIII)

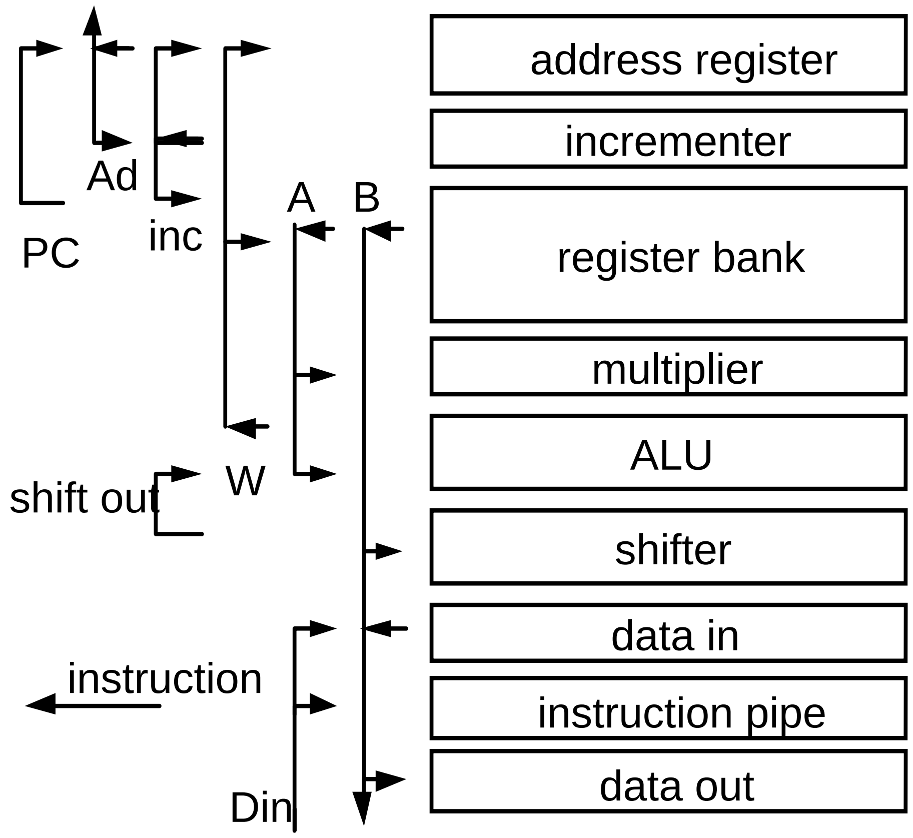
- **The register bank**
  - 1 Kbits of data: 31 general-purpose 32-bit registers
  - ARM6 register cell circuit ->
    - Works well with 5V supply
  - ARM register bank floorplan ->
  - 1/3 of total transistor count of simpler ARM cores
  - Much denser than logic functions due to higher regularity.





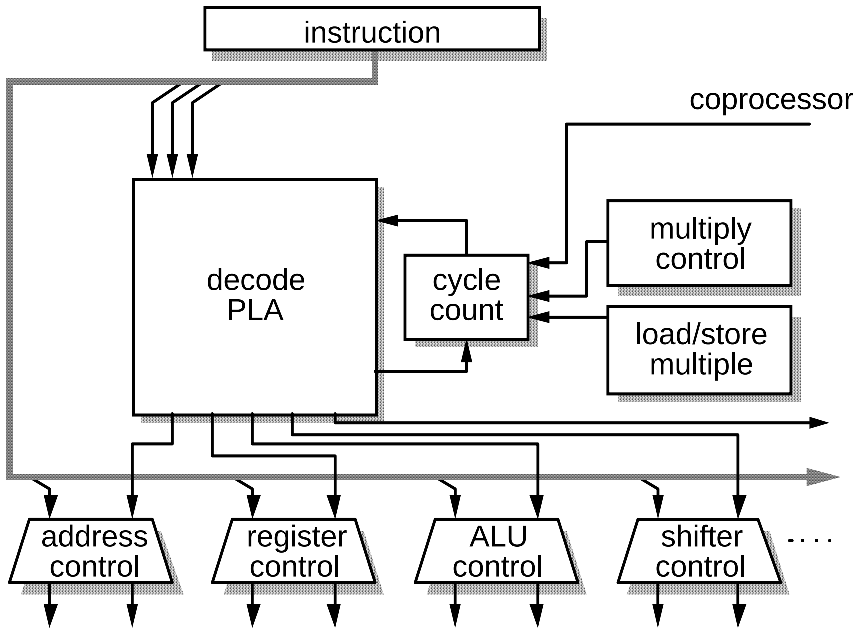
ARM Implementation (XIV)

- **Datapath layout**
  - Constant pitch per bit
  - Order of the function blocks minimize the number of additional buses passing over the more complex functions.



ARM Implementation (XV)

- **Control structures**
  - Three structural components
    - An instruction decoder PLA (programmable logic array)
      - Use some of the instruction bits & internal cycle counter
    - Distributed secondary control associated with each of the major datapath function blocks
      - Uses the class information from the main decoder PLA
      - Select other instruction bits and/or processor state infor-mation to control the datapath
    - Decentralized control units for specific instructions that take a variable number of cycles to complete
      - Main decoder PLA locks into a fixed state.



5. The ARM Coprocessor Interface

- **ARM supports**
  - A general-purpose extension of its instruction set through the addition of hardware coprocessors
  - Software emulation of these coprocessors through the undefined instruction trap
- **Coprocessor architecture**
  - Supports for up to 16 logical processors
  - Each coprocessor can have up to 16 private registers of any reasonable size; they are not limited to 32 bits
  - Coprocessors use a load-store architecture, with instructions
    - to perform internal operations on registers,
    - to load and save registers from and to memory, and
    - To move data to or from an ARM register.

The ARM Coprocessor Interface (II)

- **ARM7TDMI coprocessor interface**
  - Based on ‘bus watching’
    - The coprocessor is attached to a bus where the ARM instruction stream flows into the ARM
  - Handshake between the ARM and the coprocessor
    - Cpi’: CoProcessor Instruction (from ARM to all coprocessors)
      - ARM has identified a coprocessor instruction and wishes to execute it
    - Cpa: CoProcessor Absent (from the coprocessors to ARM)
      - Tells the ARM that there is no coprocessor present
    - Cpb: CoProcessor Busy (from the coprocessors to ARM)
      - Tells the ARM the coprocessor cannot begin executing the instruction yet.
  - Both ARM and the coprocessor must generate their respective signals autonomously.

References

- ARM organization and implementation
  - Steve Furber, “ARM System-on-chip architecture”, Second Edition, Addison Wesley, 2000.