ARM Organization and Implementation

Prepared by Dr. M. Almeer and modified by Dr. Loay Ismail

Ref: "ARM System-on-Chip Architecture", 2 Ed., by Steve Furber

1

Topics

1. 3-Stage Pipeline ARM Organization
2. 5-Stage Pipeline ARM Organization
3. ARM Instruction Execution
4. ARM Implementation

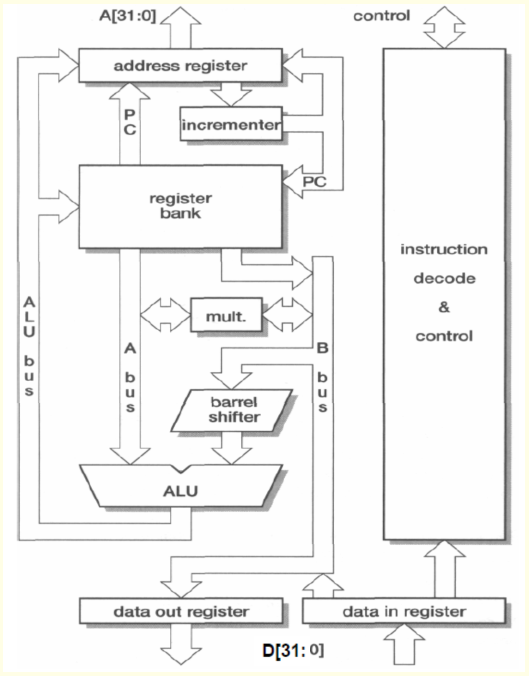
2

1. 3-Stage Pipeline ARM Organization

Principal components:

- **Register bank**: which stores the processor state. It has two read ports and one write port which can each be used to access any register, plus an additional read port and an additional write port that give special access to r15, the program counter (pc).
- **Barrel shifter**: which can shift or rotate one operand by any number of bits.
- **ALU**: which performs the arithmetic and logic functions required by the instruction set.
- **Address register and incrementer**: which select and hold all memory addresses and generate sequential addresses when required.
- **Data register**: which hold data passing to and from memory.
- **Instruction decoder and associated control logic**.

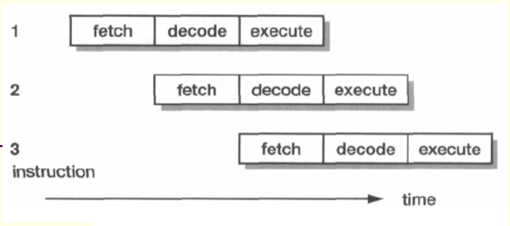
3



- In a ***single-cycle*** data processing instruction, ***two registers operands are accessed***, the value on the B bus is shifted and combined with the value on the A bus ***in the ALU***, then the result is ***written back*** into the register bank.
- The ***program counter*** value is in the ***address register***, from where it is fed into the ***incrementer***, then the incremented value is copied back into ***rl5*** in the register bank and also into the address register to be used as the address for the next instruction fetch.

4

The 3-Stage Pipeline



ARM single-cycle instruction 3-stage pipeline operation.

- Fetch
  - the instruction is fetched from memory and placed in the instruction pipeline
- 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
- Execute
  - the instruction owns the datapath; the register bank is read, an operand shifted, the ALU register generated and written back into a destination register

5

The 3-Stage Pipeline: *Single Cycle Instruction*

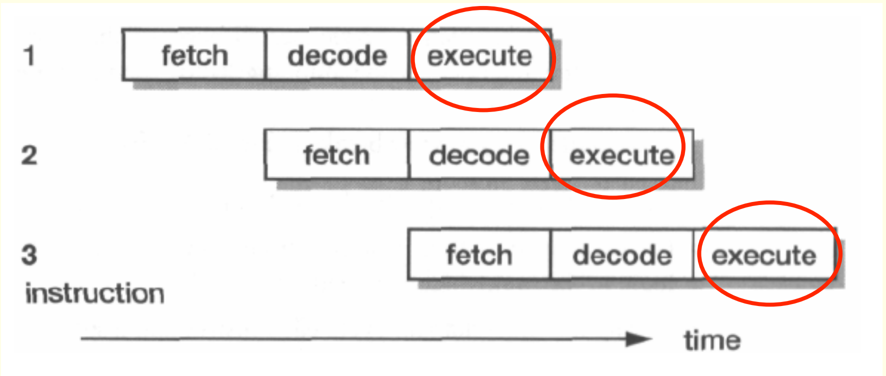


Figure 4.2 ARM single-cycle instruction 3-stage pipeline operation.

6

The 3-Stage Pipeline: *Single Cycle Instruction*

- When the processor is executing simple data processing instructions the pipeline enables **one instruction to be completed every clock cycle**.
- An individual instruction takes three clock cycles to complete, so it has a three-cycle latency, but the **throughput is one instruction per cycle.**

7

The 3-Stage Pipeline: *Multi-Cycle Instruction*

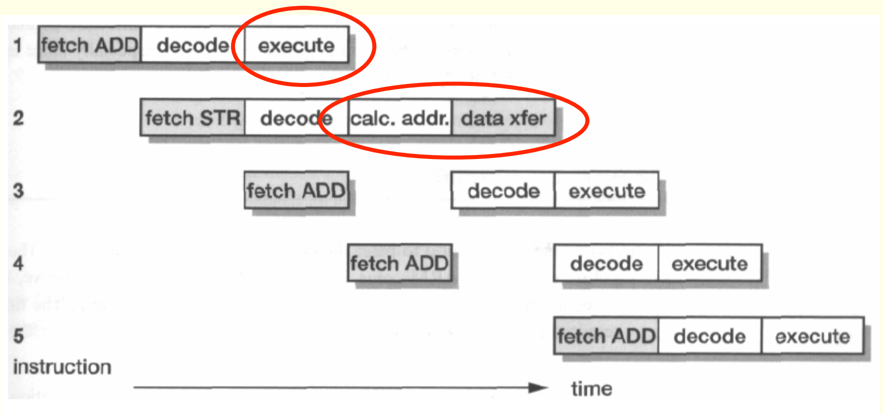


Figure 4.3 ARM multi-cycle instruction 3-stage pipeline operation.

8

The 3-Stage Pipeline: Multi-Cycle Instruction

- When a **multi-cycle instruction** is executed the **flow is less regular**, as shown in the previous slide.
- This shows a sequence of **single-cycle ADD** instructions with a **data store instruction, STR**, occurring after the first ADD.
- The **cycles that access main memory** are shown with light shading so it can be seen that memory is used in every cycle.
- The **datapath** is likewise used in every cycle, being involved in all the execute cycles, the **address calculation and the data transfer**.
- The decode logic is always generating the control signals for the datapath to use in the next cycle, so in addition to the explicit decode cycles it is also generating the control for the data transfer during the address calculation cycle of the STR.

9

How Pipeline Works?

- 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 but one (i.e. the instruction that comes after the next one)*.
- Branch instructions flush and refill the instruction pipeline.

10

Behavior of Program Counter (PC)

- One **consequence** of the **pipelined** execution model used on the ARM is that the **program counter**, which is visible to the user as r15, must **run ahead of the current instruction**.
- If instructions fetch the next instruction but one during their first execution cycle, this suggests that the PC must **point eight bytes** (**two instructi**ons) ahead of the current instruction.
- So, the **programmer who attempts to access the PC directly** through r15 must take account of the effect of the pipeline here.

11

2. 5-Stage Pipeline ARM Organization (ARM9TDMI)

- The time T_{prog} required to execute given program is given by:

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

- where
  - Ninst is the number of ARM instructions executed in the course of the program
  - CPI is the average number of clock cycles per instruction
  - fclk is the processor's clock frequency

12

2. 5-Stage Pipeline ARM Organization (ARM9TDMI)

- Since Ninst is constant for a given program there are only **two ways** to increase performance:
  - **Increase the clock rate, fclk.**
    - This requires the logic in each pipeline stage to be simplified and, therefore, the number of pipeline stages to be increased.
  - **Reduce the average number of clock cycles per instruction, CPI.**
    - This requires either that instructions which occupy more than one pipeline slot in a 3-stage pipeline ARM are re-implemented to occupy fewer slots, or that pipeline stalls caused by dependencies between instructions are reduced, or a combination of both.
    - To get a significantly better CPI, the memory system must deliver more than one value in each clock cycle either by delivering more than 32 bits per cycle from a single memory or by having separate memories for instruction and data accesses.

The 5-Stage Pipeline Organization (ARM9TDMI)

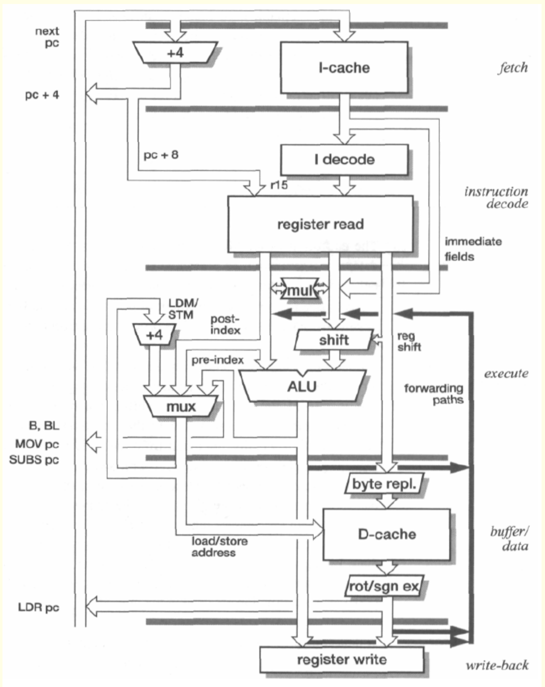


Figure 4.4 ARM9TDMI 5-stage pipeline organization.

The 5-Stage Pipeline

The 5-stage pipeline has stages:

- **Fetch**
  - The instruction is fetched from memory and placed in the instruction pipeline.
- **Decode**
  - the instruction is decoded and register operands read from the register file. There are three operand read ports in the register file, so most ARM instructions can source all their operands in one cycle.
- **Execute**
  - an operand is shifted and the ALU result generated.
  - If the instruction is a **load** or **store** the memory **address** is computed in the **ALU**.
- **Buffer/data**
  - data memory is accessed if required. Otherwise the ALU result is simply buffered for one clock cycle to give the same pipeline flow for all instructions.
- **Write-back**
  - the results generated by the instruction are **written back to the register** file, including any data loaded from memory.

15

Data Forwarding

- Instruction *Execution* in the **5-stage pipeline** is spread across **3 pipeline stages** (Execute, Buffer/Data, and Write-back) in the 5-stage pipeline.
- The **only one way to solve data dependencies** without stalling the 5-stage pipeline is to introduce *forwarding paths* (as shown in the 5-stage pipeline organization).
- ***Data dependencies*** arise when an instruction ***needs to use the result of one of its predecessors*** before that result has returned to the register file.
- ***Forwarding paths*** allow results to be passed between stages as soon as they are available. The **5-stage ARM pipeline** requires each of the **3 source operands** to be forwarded from any of 3 intermediate result registers.

16

Data Forwarding

- There is a case where, **even with forwarding, it is not possible to avoid a pipeline stall**.
- Consider the following code sequence:

  LDR r_N , [. .]   ; Load r_N from somewhere
  ADD r2, r1, r_N ; and use it immediately

- The processor **cannot avoid a one-cycle stall** as the value loaded into ***r_N*** only enters the processor at the end of the buffer/data stage and it is needed by the following instruction at the start of the execute stage.
- The **only way to avoid this stal**l is to encourage the **compiler** (or assembly language **programmer**) **not to put a dependent instr**uction immediately after a load instruction.

17

Data Forwarding

- Example:
- The following sequence of instructions:

  LDR r3, [r2, r1, LSL #2]   ; r3 := Mem[ ]
  ADD r5, r5, **r3**, LSL #3   ; r5 := r5 + r3 x 8
  ADD r7, r8, r9   ; r7 := r8 + r9

- Can be replaced by this sequence:

  LDR r3, [r2, r1, LSL #2]   ; r3 := Mem[ ]
  ADD r7, r8, r9   ; r7 := r8 + r9
  ADD r5, r5, **r3**, LSL #3   ; r5 := r5 + r3 x 8

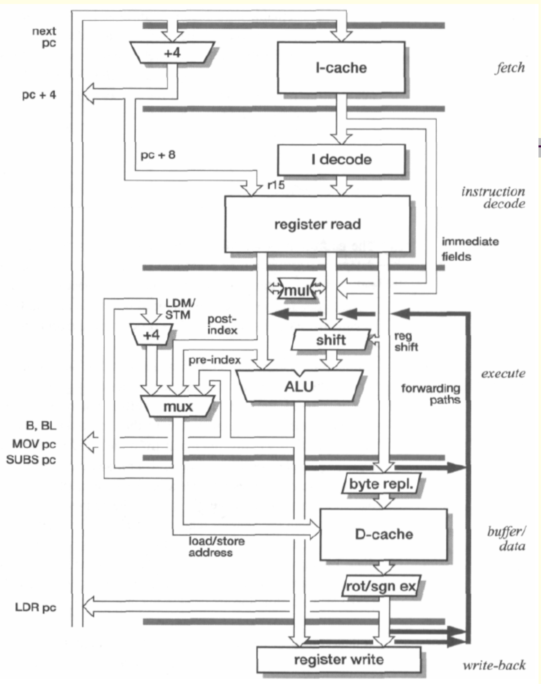


Figure 4.4 ARM9TDMI 5-stage pipeline organization.

18

PC Generation

- **3-stage pipeline**
  - PC behavior:

  operands are read in *execution stage*
  r15 = PC + 8
- **5-stage pipeline**
  - operands are read in *decode stage* and r15 = PC + 4
  - incompatibilities between 3-stage and 5-stage implementations
  - to avoid this, 5-stage pipeline ARMs emulate the behavior of the older 3-stage designs
  - PC value from the fetch stage is fed directly to the register file in the decode stage, bypassing the pipeline register between the two stages. PC+4 for the next instruction is equal to PC+8 for the current instruction.

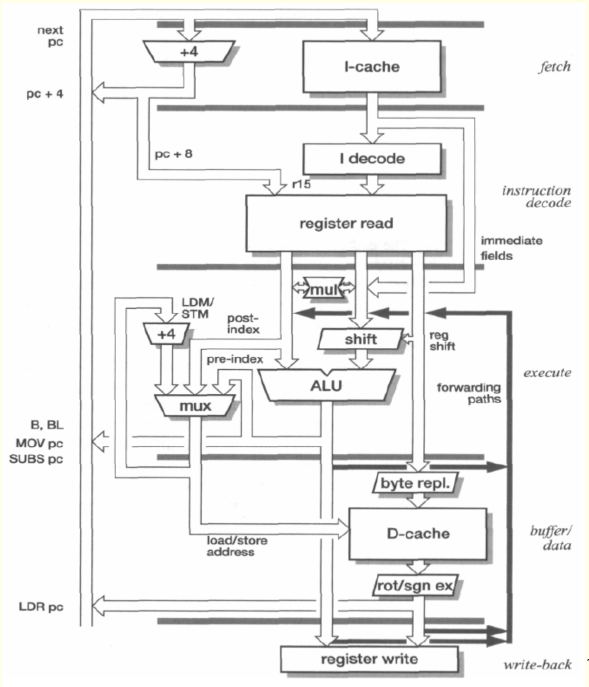


Figure 4.4 ARM9TDMI 5-stage pipeline organization.

19

3. ARM Instruction Execution

- The execution of an ARM instruction can best be understood by using an annotated version of the 3-stage pipeline organization diagram, omitting the control logic section, and highlighting the active buses to show the movement of operands around the various units in the processor.

20

Data Processing Instruction Datapath Activity

- **Reg-Reg**
  - Rd = Rn op Rm
  - PC = AR + 4
  - AR = AR + 4
- **Reg-Imm**
  - Rd = Rn op Imm
  - PC = AR + 4
  - AR = AR + 4

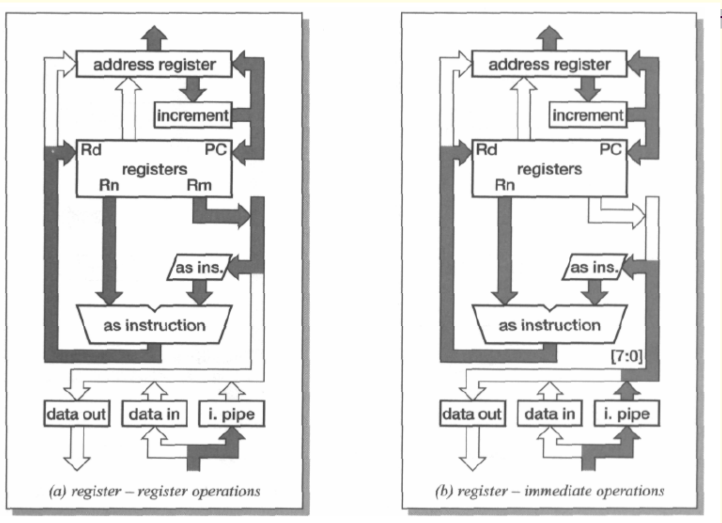


Figure 4.5 Data processing instruction datapath activity.

21

STR (Store Register) Datapath Activity

- **Compute address**
  - PC = AR + 4
  - AR = Rn op Disp
- **Store data**
  - mem[AR] = Rd<x:y>
  - If autoindexing
    => Rn = Rn +/- 4
  - AR = PC

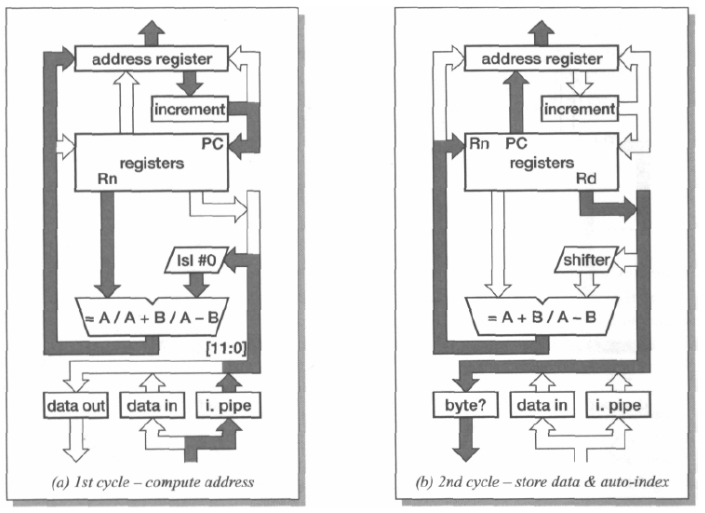


Figure 4.6 SIR (store register) datapath activity.

22

Cycles of a branch instruction

- **First Cycle**: Compute target address
  - AR = PC + Disp,lsl #2
- **Second Cycle**: Save return address (if required)
  - r14 = PC
  - AR = AR + 4
- **Third cycle**:
  - Do a small correction to the value stored in the link register in order that it points directly at the instruction which follows the branch (i.e. it should point to ***PC + 4*** and not ***PC + 8***

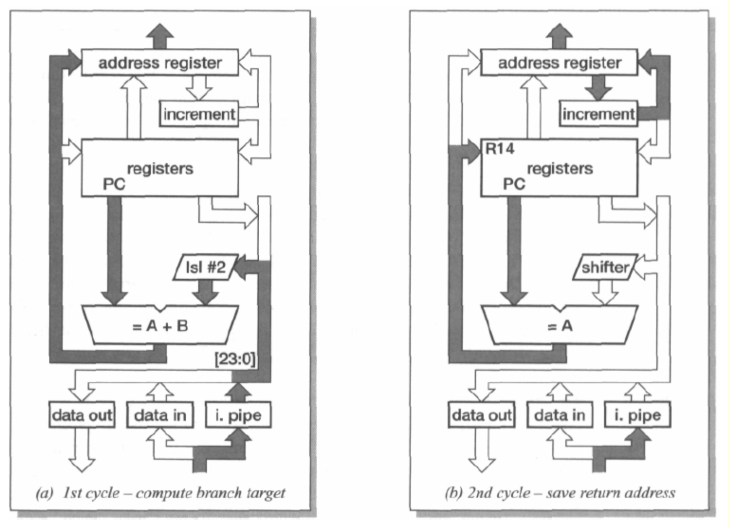


Figure 4.7 The first two (of three) cycles of a branch instruction.

4. ARM Implementation

- The design is divided into:
  - **Datapath:** described in Register Transfer Level (RTL) notation
  - **Control Unit:** Finite State Machine (FSM)

Clocking Scheme

- Most ARMs do not operate on edge-sensitive registers
- Instead the design is based around 2-phase non-overlapping clocks which are generated internally from a single input clock signal
- Data movement is controlled by passing the data alternately through latches which are open during phase 1 and latches which are open during phase 2.
- The non-overlapping property of the phase 1 and phase 2 clocks ensures that there are no race conditions in the circuit.

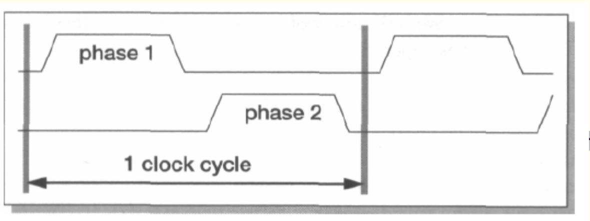
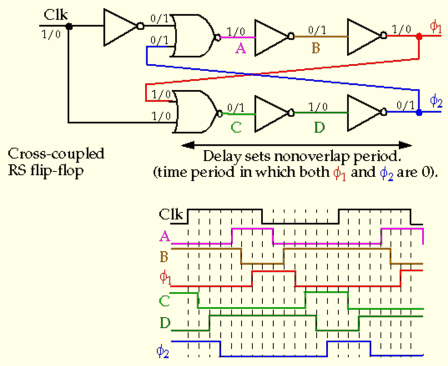


Figure 4.8 2-phase non-overlapping clock scheme.


Clk
1/0
0/1
A
B
C
D
ϕ_1
ϕ_2
Cross-coupled RS flip-flop
Delay sets nonoverlap period.
(time period in which both ϕ_1 and ϕ_2 are 0).

Data path Timing

- **Register read**
  - Register read buses – dynamic, precharged during phase 2
  - During phase 1 selected registers discharge the read buses which become valid early in phase 1
- **Shift operation**
  - During phase 1, second operand passes through barrel shifter
- **ALU operation**
  - ALU has input latches which are open in phase 1, allowing the operands to begin combining in ALU as soon as they are valid, but they close at the end of phase 1 so that the phase 2 precharge does not get through to the ALU
  - ALU processes the operands during the phase 2, producing the valid output towards the end of the phase, and the result is latched in the destination register at the end of phase 2

Datapath Timing

- The minimum datapath cycle time is therefore the sum of:
  - the register read time;
  - the shifter delay;
  - the ALU delay;
    - (dominates and highly variable: Logic operations are fast while arithmetic are slow due to propagation delays)
  - the register write set-up time;
  - the phase 2 to phase 1 non-overlap time.

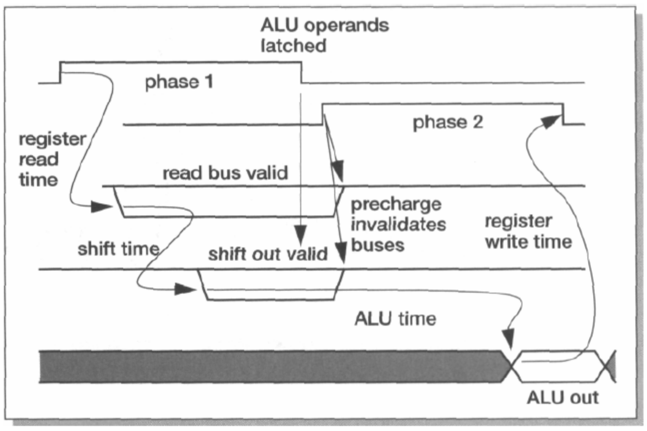


Figure 4.9 ARM datapath timing (3-stage pipeline).

27

Adder Design (Original Ripple Carry Adder)

- Using an AND-OR-INVERT gate for the carry logic and alternating AND/ OR logic so that:
  - even bits use the circuit shown
  - odd bits use the dual circuit with inverted inputs and outputs and AND and OR gates swapped around
- Worst-case carry path is 32 gates long.

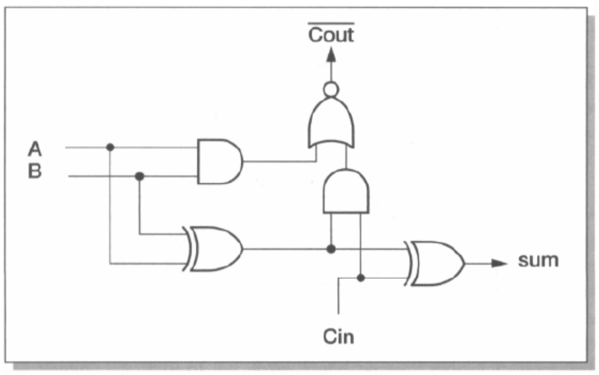


Figure 4.10 The original ARM1 ripple-carry adder circuit.

28

Adder Design (4-bit Carry Look-ahead)

- Carry generate (G) and Carry propagate (P) signals control the 4-bit carry-out.
- Where
  - Cout[3] = (G + P.Cin[0])
- Worst case carry propagate path length is reduced to 8 gate delays.
  - Using merged AND-OR-INVERT gates and alternating AND/OR logic

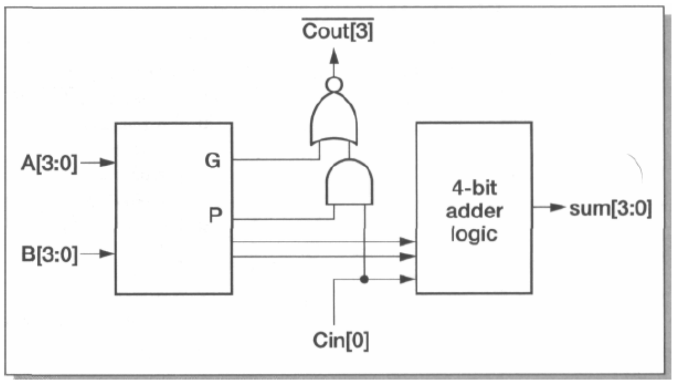


Figure 4.11 The ARM2 4-bit carry look-ahead scheme.

29

ALU Function

- ALU functions must cover the full set of data operations defined by the instruction set, including:
  - Add and subtract
  - Address computations for memory transfers,
  - Branch calculations,
  - Bit-wise logical functions,
  - and so on.

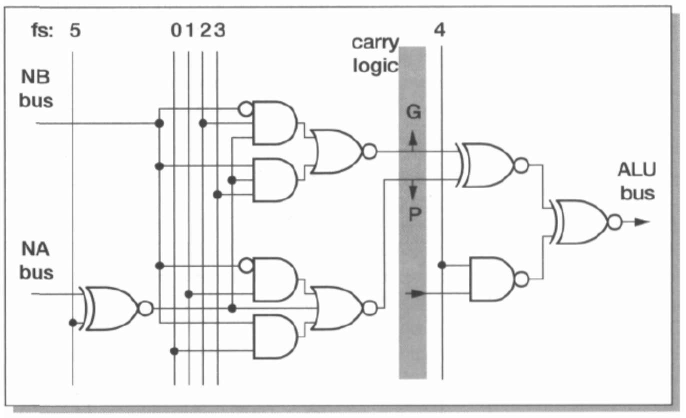


Figure 4.12 The ARM2 ALU logic for one result bit.

30

ALU Function

- The set of functions generated by this ALU and the associated values of the ALU function selects are listed in the shown table.

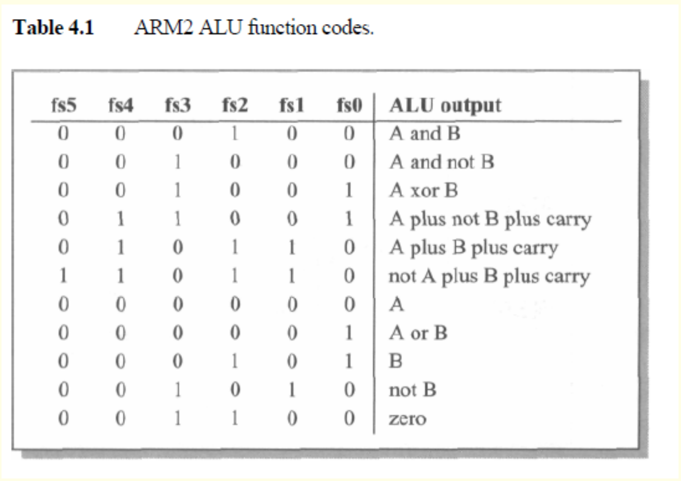


Table 4.1 ARM2 ALU function codes.

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

31

The Cross-Bar Barrel Shifter

- The shifter performance is critical since the shift time contributes directly to the datapath cycle time
- In order to minimize the delay through the shifter, a cross-bar switch matrix is used to steer each input to the appropriate output.
- The principle of the cross-bar switch is illustrated in the figure, where a 4 x 4 matrix is shown. (The ARM processors use a 32 x 32 matrix.)

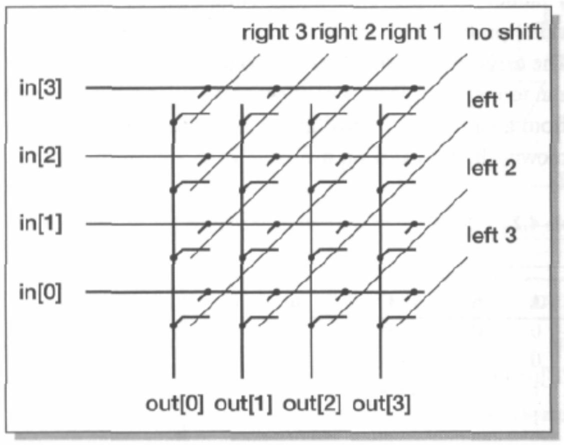


Figure 4.15 The cross-bar switch barrel shifter principle.

32

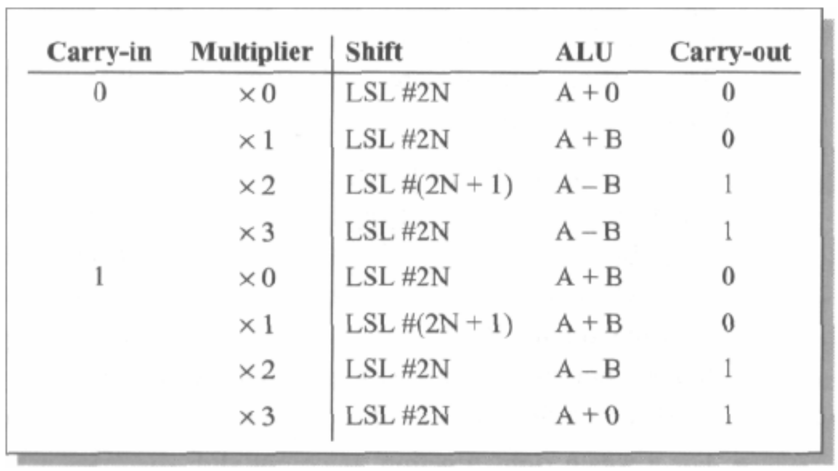
The Cross-Bar Barrel Shifter

- Each input is connected to each output through a switch.
- Pre-charged dynamic logic is used, as it is on the ARM datapaths, where each switch can be implemented as a single NMOS transistor.
- Precharging sets all outputs to logic 0, so those which are not connected to any input during switching remain at 0 giving the zero filling required by the shift semantics
- For a rotate right function, the right shift diagonal is enabled together with the complementary left shift diagonal. For example, on the 4-bit matrix rotate right one bit is implemented using the 'right 1' and the 'left 3' (3 = 4 - 1) diagonals.
- Arithmetic shift right uses sign-extension rather than zero-fill for the unconnected output bits. Separate logic is used to decode the shift amount and discharge those outputs appropriately.

33

The 2-bit Multiplication Algorithm, Nth Cycle

Table 4.3 The 2-bit multiplication algorithm, Nth cycle.



Carry-in	Multiplier	Shift	ALU	Carry-out
0	×0	LSL #2N	A + 0	0
	×1	LSL #2N	A + B	0
	×2	LSL #(2N + 1)	A – B	1
	×3	LSL #2N	A – B	1
1	×0	LSL #2N	A + B	0
	×1	LSL #(2N + 1)	A + B	0
	×2	LSL #2N	A – B	1
	×3	LSL #2N	A + 0	1

34

Carry Propagate and Carry Save Adder Structures

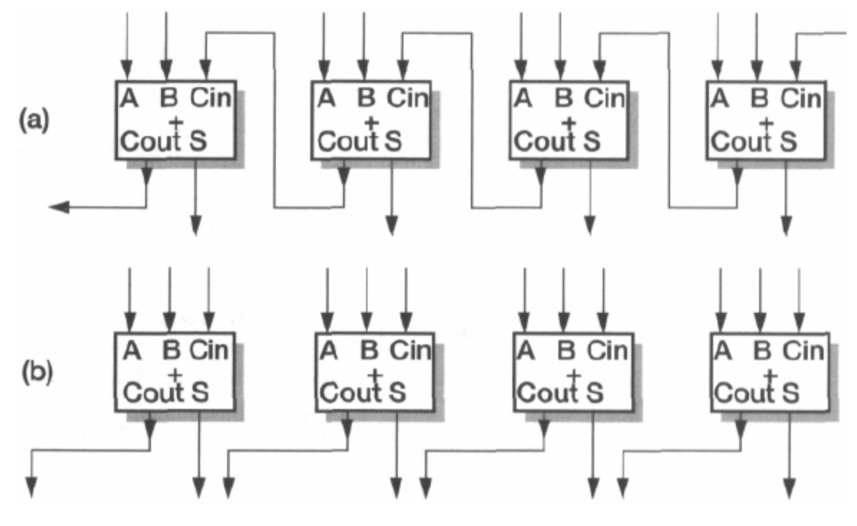


Figure 4.16 Carry-propagate (a) and carry-save (b) adder structures.

35

ARM High Speed Multiplier Organization

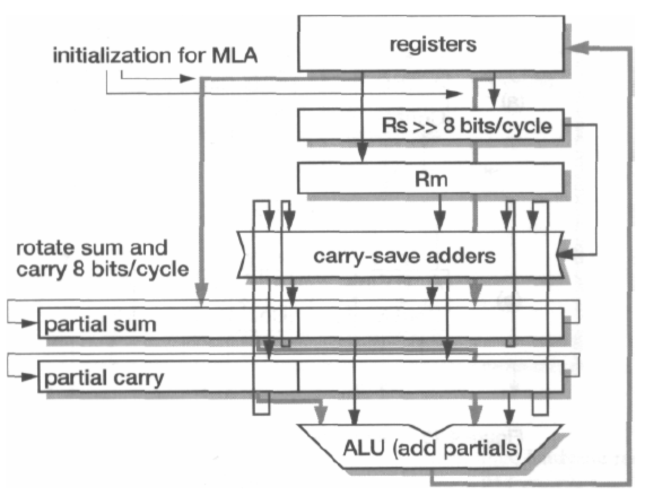


Figure 4.17 ARM high-speed multiplier organization.

36

ARM6 Register Cell Circuit

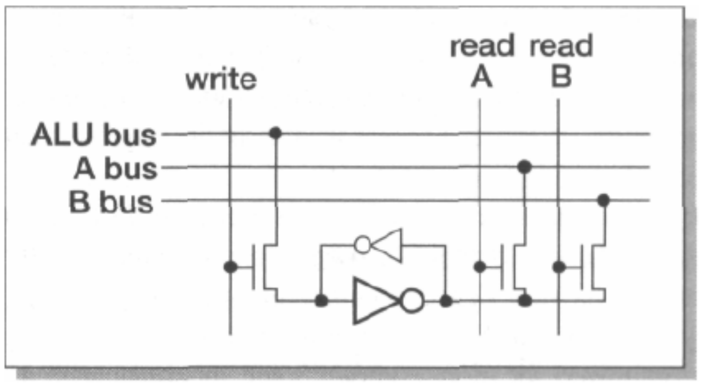


Figure 4.18 ARM6 register cell circuit.

37

ARM Register Ba Floorplan

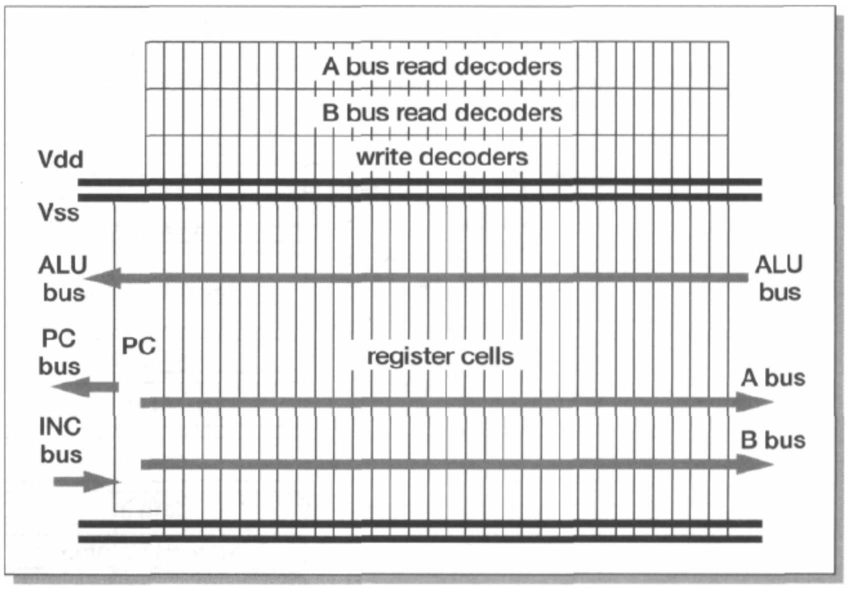


Figure 4.19 ARM register bank floorplan.

38

Data-path Layout

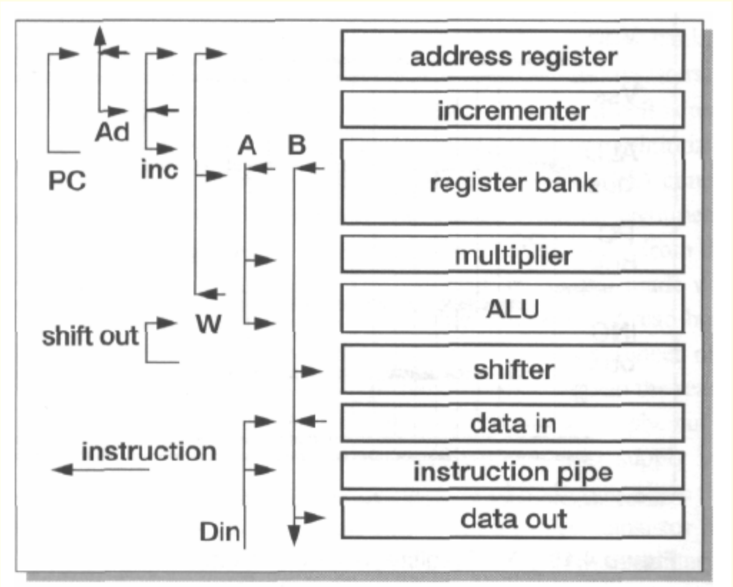


Figure 4.20 ARM core datapath buses.

39

ARM Control Logic Structure

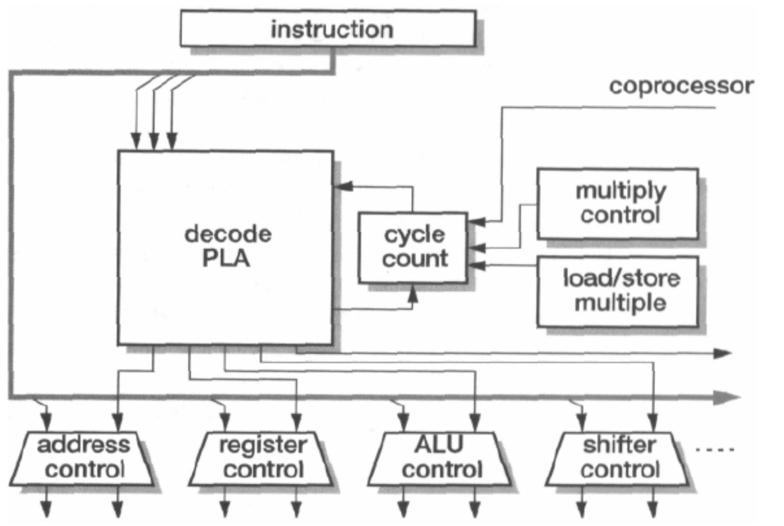


Figure 4.21 ARM control logic structure.

40