29.3 Decode, Execute Hardware

Decoding an instruction means interpreting the operands and accessing the appropriate values which may be found as part of the instruction (i.e., in the instruction register) or in internal memory. Therefore, a component to provide internal memory is introduced by using a register file..

Having provided hardware for retrieving the operands, to execute the instruction requires an arithmetic logic unit with sufficient functionality to perform any arithmetic or logic computation specified by the instruction set, including the computation of an effective address if the instruction used bas+displacement mode.

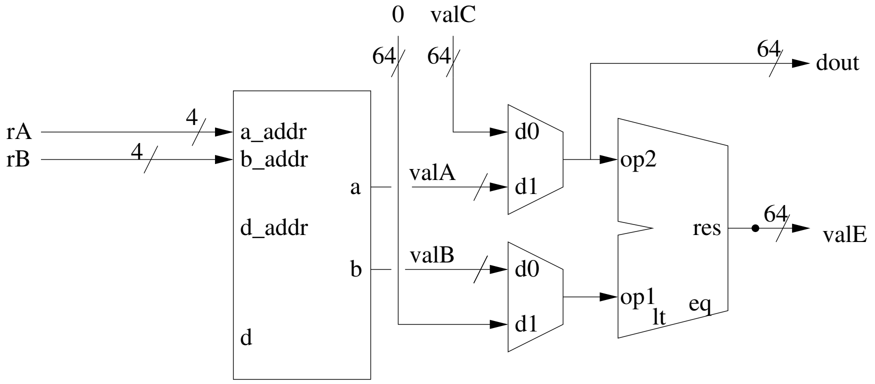
The component used to implement this stage of the instruction execution algorithm are:

$2^4 \times 64$ **Register File** : to provide the internal memory for the CPU as per the original instruction set architecture specifications.

2×64 Multiplexer : to select the first operand for the ALU. From the summary of register transfer statements of the instruction execution algorithm, the first operand should be either `R[rB]` or 0.

2×64 Multiplexer : to select the second operand for the ALU. This will be `R[rA]` when an arithmetic/logic instruction was decoded, and `IR(valC)` when an effective address must be calculated because base+displacement mode was used by the decoded instruction.

ALU : to meet the computational requirements indicated by the register transfer statements of the instruction execution algorithm.



29.4 Writeback Hardware

The result produced by the ALU must now be delivered to the appropriate destination. If the instruction executed is an arithmetic/logic instruction then the result should be stored back in internal memory at `R[rB]`. If the result is an effective address then it should be stored in `ACTR` to provide the address for storage of a data value from external memory `M` in `R[rA]` or the retrieval of a value in `M` and its storage in `R[rA]`. As well the results of a comparison of the two operands of the ALU need to be stored.

Counter Register ACTR : To hold the address of the first of 8 bytes of external memory where an 8-byte value will be retireved or stored. BY incrementing the counter from 0 to 7, the address of each byte can be determined.

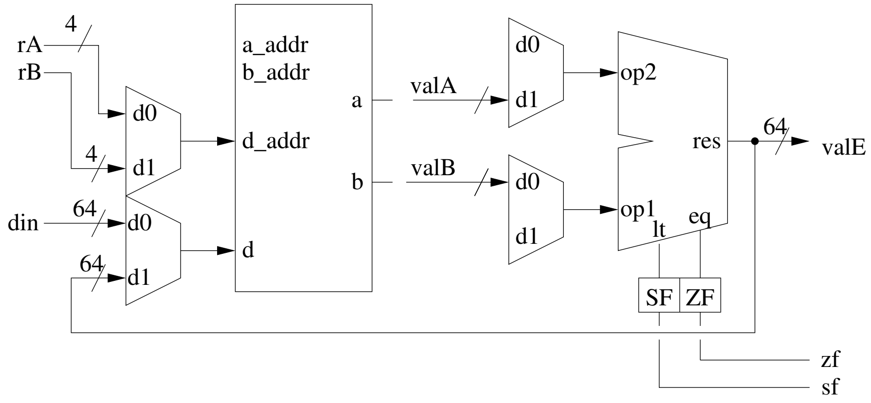
2×64 Multiplexer : There are two ways the internal memory may be updated:

1. The value computed by the ALU is stored in a register of internal memory.
2. A value retrieved from external memory, `M`, is stored in a register of internal memory.

2×4 Multiplexer : to select which register, `rA` or `rB`, should be the destination for a value to be stored in the register file. Note that the `mov` instruction (opcode 42) stores a value in `R[rA]` but the arithmetic/logic instructions store the result in `R[rB]`.

1-bit storage register, SF : Set to 1 if the first operand of the ALU is less than the second operand.

1-bit storage regiset, ZF : Set to 1 if the first operand of the ALU equals the second operand.



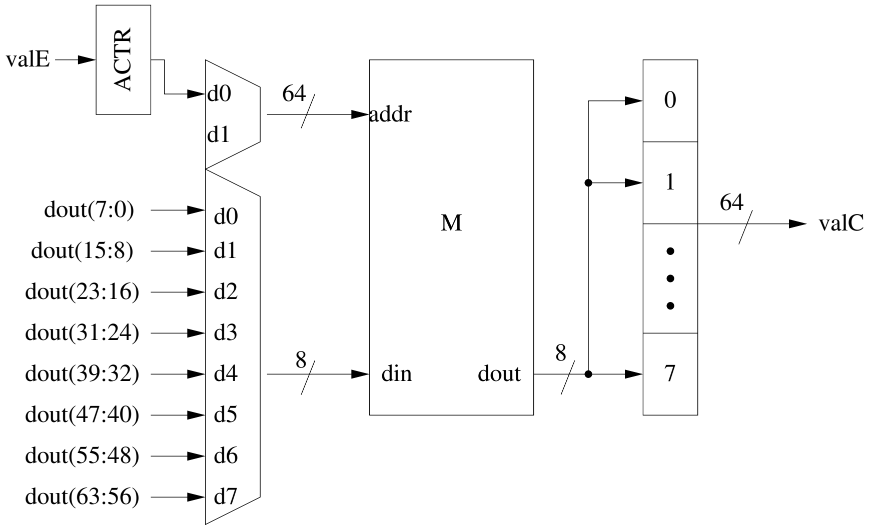
29.5 Memory Hardware (Data Retrieve & Store)

Because the CPU internal memory consists of 64-bit words and the external memory consists of 8-bit words, transferring data between the two memories requires hardware to pack 1-byte values into 8-byte values and unpack 8-byte values into 1-byte values:

8×8 Multiplexer : A 64-bit bus is partitioned into eight 8-bit buses, each connected to a different 8-bit port of the MUX. By selecting ports 0 through 7 sequentially, the 8-byte value on the 64-bit bus can be delivered 1-byte at a time to the data input port of `M`. At the same time, the ACTR register is incremented each time the next MUX port is selected so that the bytes are stored sequentially in memory.

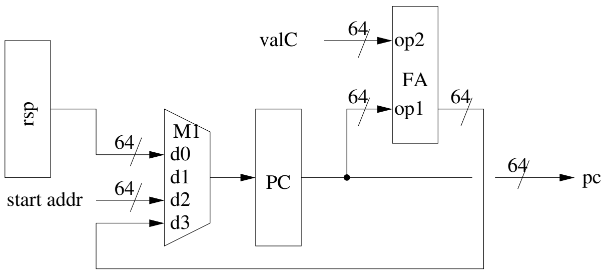
Eight 8-bit Storage Registers : For retrieval of a data value from `M`, the ACTR register is incremented and at the same time, each of the eight 8-bit registers is enabled, one at a time, allowing eight 8-bit values to be stored. The eight 8-bi register outputs are then grouped into a single 64-bit bus for delivery to the the multiplexer that controls which value will be stored in internal memory.

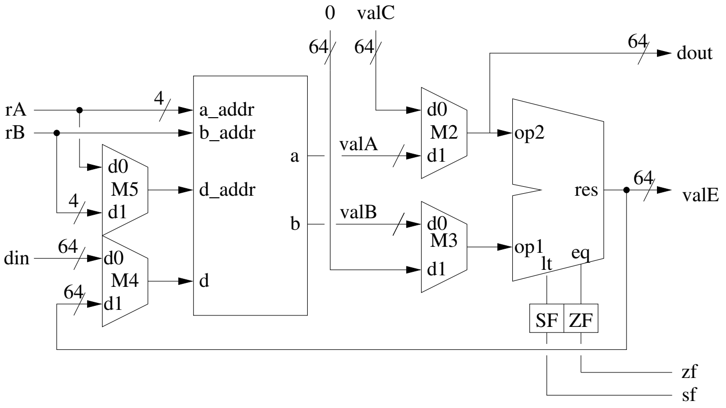
.

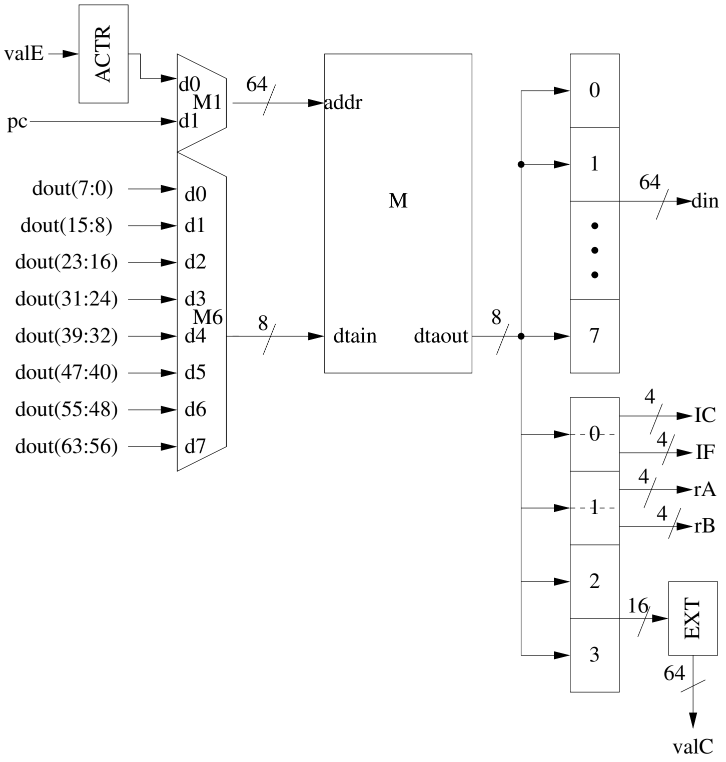


29.6 CPU Datapath Summary

The subsystems just developed can be merged into a single schematic that reflects the architecture of the CPU datapath as shown on the next page.


valC
64
op2
FA
64
op1
64
rsp
64
M1
d0
d1
PC
64
pc
start addr
64
d2
d3


0
valC
64
64
64
dout
rA
4
a_addr
rB
b_addr
d0
M2
valA
d1
op2
a
d0
M5
d_addr
4
d1
res
64
valE
b
valB
d0
M3
d1
op1
lt
eq
din
64
d0
M4
d
64
d1
SF
ZF
zf
sf


valE
ACTR
d0
M1
64
addr
pc
d1
M
0
1
64
din
7
dout(7:0)
d0
dout(15:8)
d1
dout(23:16)
d2
dout(31:24)
d3
M6
dout(39:32)
d4
8
dtain
dtaout
8
dout(47:40)
d5
dout(55:48)
d6
dout(63:56)
d7
0
4
IC
4
IF
1
4
rA
4
rB
2
16
EXT
3
64
valC

30 CPU PERFORMANCE

Given a CPU design, the instruction "execution" time can be measured in clock "cycles"; that is, the number of clock periods required to fetch the instruction from memory and then execute it. To express the execution time in seconds, The number of cycles is multiplied by the clock cycle period.

30.1 Unfolded Data Flow Diagrams

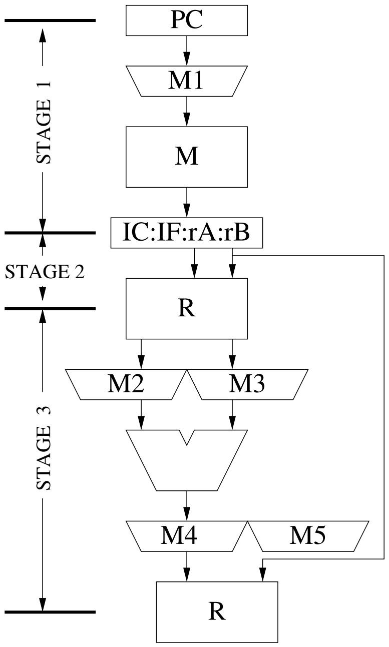
A systematic way to analyze the effect of propagation delay on the execution of each instruction is to construct a diagram of the sequence of components that traces the signal transfer behavior when each instruction is fetched and executed. Such a diagram is called the *unfolded data flow (UDF) diagram* for the given machine instruction and identifies the sequence of components required to support the fetch and execute phases of that instruction.

For each instruction, to construct the UDF for that instruction, the CPU datapath is used to trace such a sequence, beginning with the program counter (PC), since that register specifies the location of the instruction in memory that is about to be retrieved.

For example, consider the datapath of the example CPU that has been developed.

The UDF diagram for the ADD machine instruction (opcode `X"20"`) is provided on the next page.

Example: add Rs, Rd



30.2 Computing the clock cycle period

1. Determine the propagation delay, t$_{pd}$, of each component.
2. Divide the UDF into stages as defined by synchronous components.
3. For each instruction, determine the propagation delay of each stage:
    - If there is only only one data flow path, sum the propagation delays of all components, beginning with the register component defining the stage, but not including the register component defining the next stage.
    - If there is more than one dataflow path, sum the propagation delays of all paths and choose the maximum.
4. The clock cycle period is the minimum integral value greater than the maximum propagation delay of any stage.

Example: Consider the UDF diagram for the add instruction given above.

For each component, a propagation delay for that component can been assigned. Specifically, suppose the components in the datapath have the following propagation delays:

- Register File: 15 ns
- Registers: 5 ns
- Multiplexers: 10 ns
- ALU: 40 ns
- Memory: 100 ns
- EXT: 10 ns

The propagation delay of each stage is computed by summing the propagation delays of all components in a stage beginning with the starting register or register file, but not including the final register or register file. If there is more than one "path" through the stage, then the maximum value of all paths defines the propagation delay. For the UDF diagram for the ADD instruction above, the following times are obtained:

Stage 1 : 5 + 10 + 100 = 115 ns

stage 2 : 5 ns

stage 3 : max(15 + 10 + 40 + 10, 15 + 10 + 40 +10, 5) = 75 ns

The miminum clock cycle period must be long enough to accommmodate both the propagation delay and the set-up time of the final register. The "set-up" time is the time before a register is enabled that the input must be available on the input bus. For the example, the minimum clock period is 116 ns, assuming a set-up time of 1 ns.

30.3 Improving CPU Performance

Since execution time is a function of the clock period and the number of stages in the UDF diagram, several strategies can be considered:

Clock period reduction : Choosing the best possible clock period is an obvious approach. Determining the minimum clock period is described above. So this value should be chosen as the clock period if the clock currently has a longer period.

Faster memory access : The effect of the von Neumann bottleneck is evident from the UDF diagram above. Therefore it is important to seek improvements in memory technologies that will reduce the propagation delay of this component.

Faster CPU components : If the propagation delay of memory can be reduced, then it may no longer determine the "critical stage" of the UDF diagram; that is, some other stage may have a longer propagation delay. In this case, it may be possible to gain further improvements by replacing existing components with equivalent ones that have a shorter propagation delay.

Modification of the CPU architecture : Redesigning the CPU architecture will change the UDF diagram for each instruction. Such a change may improve the "worst case" propagation delay. One simple approach is to introduce an additional register into the architecture that partitions a stage with a long propagation delay into two stages each with reduced propagation delays, at the cost of adding an additional stage.

Parallel processing of instructions (Pipelining) : Redesigning the CPU so that more than one instruction can be executed simultaneously then the "average execution time" can be reduced significantly:

Average execution time per instruction:

$$= \frac{\text{total time for execution of all instructions}}{\text{number of instructions}}$$

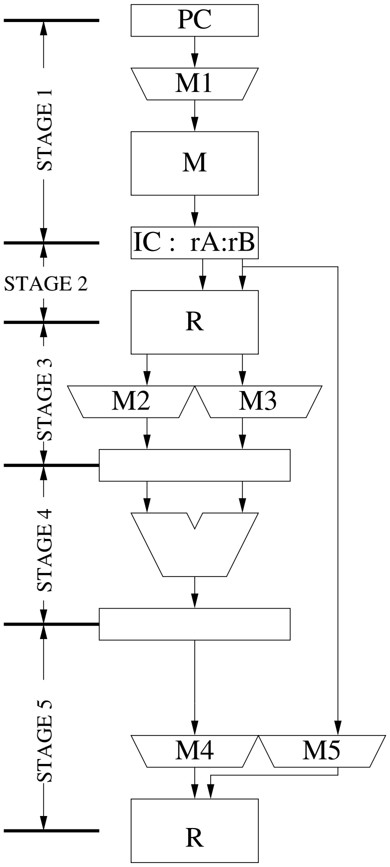
Modify CPU - Memory Interface (Block Data Transfer) : Retrieving more than one location of memory during a memory access will reduce the average memory access time. A"block" of memory is a sequence of consecutive locations of memory that are retrieved during a single memory access.

Hierarchical memory architecture (Cache Memory) : By providing faster storage than that provided by the memory technology used for external memory, recent memory retrievals can be retained in faster memory for subsequent faster retrieval if the same locations must be accessed again.

To illustrate with the ADD statement example, two strategies can be employed to the example:

1. A fast memory will be used for M, whose propagation delay is 20 ns.
2. Two registers will be used to partition stage 3 into three stages

The revised UDF diagram for the ADD instruction is:



Stage 1 : 5 + 10 + 100 = 115 ns

stage 2 : 5 ns

stage 3 : max(15 + 10, 15 + 10) = 25 ns

stage 4 : 5 + 40 = 45 ns

stage 5 : max(5 + 10, 5) = 15 ns

So the minimum clock cycle period for the ADD instruciton in this revised archtiecture is 45 + 1 = 46 ns, assuming a set-up time of 1 ns.

Note that the benefit of the shorter clock cycle period is offset by the increased number of stages. For the two examples shown, the execution time of the ADD instruction is:

Version 1 : 116 ns × 3 stages = 348 ns.

Version 2 : 46 ns × 5 stages = 230 ns.

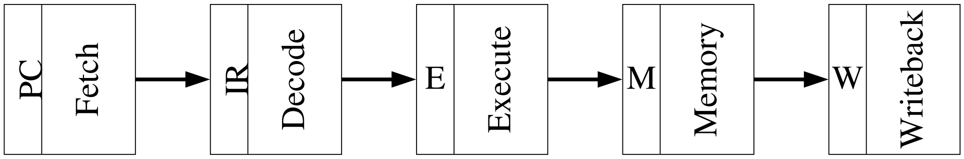
31 STAGED CPU ARCHITECTURES

Partitioning a CPU architecture into stages results in a reduction in the clock period and a possible improvement in execution time. However, it can also partition the CPU into functional units that can be executed sequentially. If information is retained at each stage then it is possible to have each stage function independently and so provide the opportunity for parallel execution.

For the purpose of retaining the required information from each stage, the initial register of each stage is increased in size so that it can hold a copy of all the required values. Such a register is called a "pipeline register."

31.1 5-Stage Machine

A CPU datapath that is partitioned into n stages by a set of pipline registers is called an "n-stage machine," A 5-stage machine is used in many CPU architectures, including the x86-64. The basic organization of the 5-stage machine is:



To configure the CPU datapath described previously to a 5-stage pipelined architecture, it is necessary to identify what must be stored in each pipeline. This can be determined by referring back to the instruction execution algorithm and observing which registers and buses (i.e., variables) are required at each stage:

Pipeline register PC : The address of the next instruction

Pipeline register IR : The instruction components: IC : IF : rA : rB : valC

Pipeline register E : The values of IC, IF, rA, rB, valA, valB, valC

Pipeline register M : The values of IC, IF, rA, rB, valE, din

Pipeline register W : THe values of IC, IF, rB, sf, zf, valE