4.3.1) clock cycle time은 critical path를 지나는 경로이다. 그림의 total latency중에서도 critical path를 찾아야하는 local 문제이다.
load 의 Data path는 생각해보면 Inst. Memory → Register file (명령어fetch) (read reg)
→ MUX → ALU → Data Memory → Mux
(get addr) (addr+reg) (get Data) (Reg write)
위와같다. 이 경로의 latency를 계산하면 answer 400ps+200ps+30ps+120ps+350ps+30ps=1130ps이다.
그리고 Improvement 후에는 ALU에 300 ps의 latency가 추가되고 critical path는 ALU를 사용하지않음
total latency는 1130 ps + 200ps = 1430 ps이다.

4.3.2) speedup은 critical path를 improvement 전과 후 latency를 구하고
Instruction을 줄여 프로그램처리속도를 향상시킨다.
Improvement 에서 5%의 Instruction이 줄었고 4.3.1)에서 보았듯이 latency가 늘어났다. 초기의 속도를 x라하면
$x \times \frac{100}{95} \times \frac{1130}{1430} = 0.8317x$ 이다. 이로서 약 17%의 속도저하가 발생했음을 알수 있다.

CA HW#2

4.3 When processor designers consider a possible improvement to the processor datapath, the decision usually depends on the cost/performance trade-off. In the following three problems, assume that we are starting with a datapath from Figure 4.2, where I-Mem, Add, Mux, ALU, Regs, D-Mem, and Control blocks have latencies of 400 ps, 100 ps, 30 ps, 120 ps, 200 ps, 350 ps, and 100 ps, respectively, and costs of 1000, 30, 10, 100, 200, 2000, and 500, respectively.

Consider the addition of a multiplier to the ALU. This addition will add 300 ps to the latency of the ALU and will add a cost of 600 to the ALU. The result will be 5% fewer instructions executed since we will no longer need to emulate the MUL instruction.

4.3.1 [10] <§4.1> What is the clock cycle time with and without this improvement?

4.3.2 [10] <§4.1> What is the speedup achieved by adding this improvement?

4.3.3 [10] <§4.1> Compare the cost/performance ratio with and without this improvement.

4.3.3) cost는 모든 Component를 생각해야한다. 그림에서 I-Mem, 2x Add, 3x MUX, ALU, Reg, D-Mem, Control Block 이렇게 총 10개의 Component가있다. 모든 Component의 cost를 계산하면 1000 + 2x30 + 3x10 + 100 + 200 + 2000 + 500 을 3890 원이다. Improvement 후에 ALU cost에 600이 추가되었으므로 4490이다. 이것을사용해 cost/performance 식을세우면
Branch $\frac{4490}{3890} \times \frac{100}{83} = 1.3906\cdots$ 로 Improvement 전보다 1.39배나 느린 프로세서가 된다.

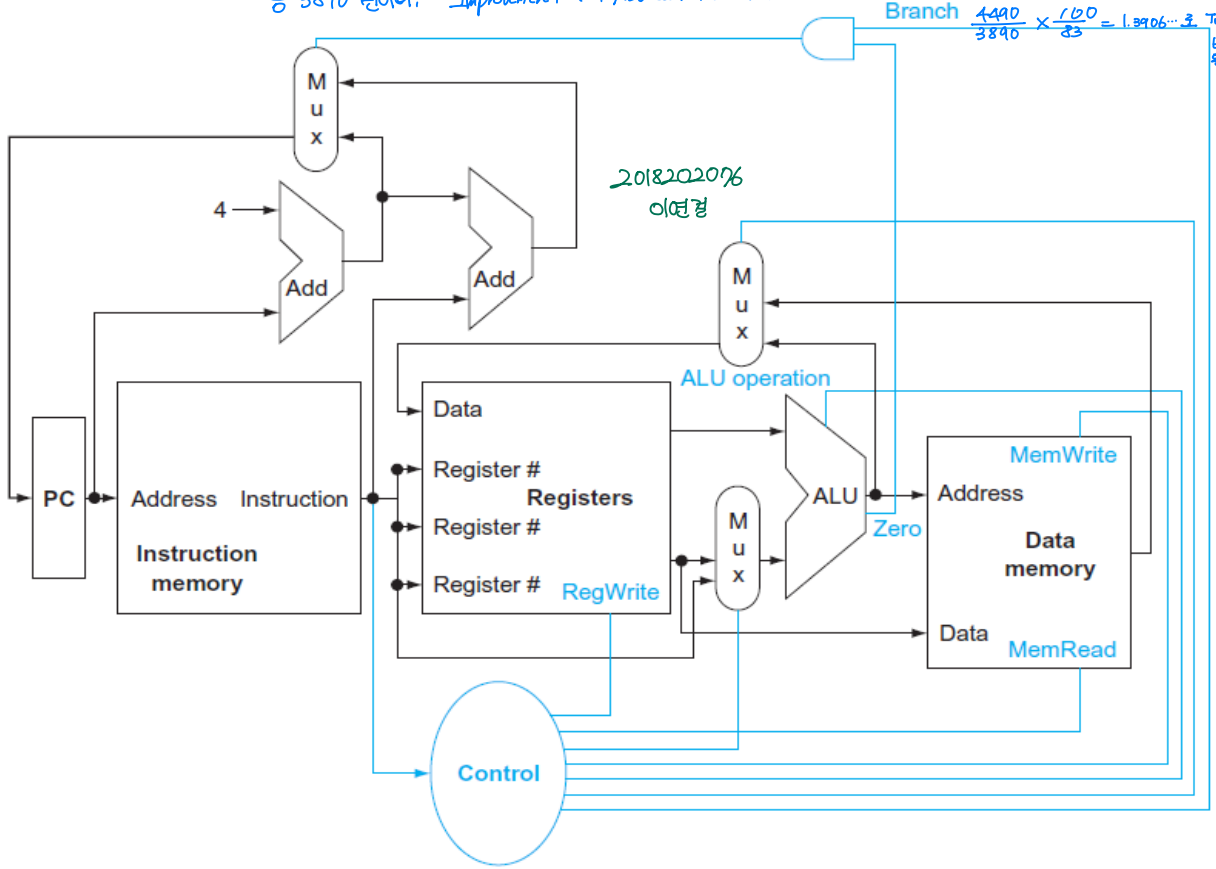

FIGURE 4.2 The basic implementation of the MIPS subset, including the necessary multiplexors and control lines. The top multiplexor ("Mux") controls what value replaces the PC (PC + 4 or the branch destination address); the multiplexor is controlled by the gate that "ANDs" together the Zero output of the ALU and a control signal that indicates that the instruction is a branch. The middle multiplexor, whose output returns to the register file, is used to steer the output of the ALU (in the case of an arithmetic-logical instruction) or the output of the data memory (in the case of a load) for writing into the register file. Finally, the bottommost multiplexor is used to determine whether the second ALU input is from the registers (for an arithmetic-logical instruction or a branch) or from the offset field of the instruction (for a load or store). The added control lines are straightforward and determine the operation performed at the ALU, whether the data memory should read or write, and whether the registers should perform a write operation. The control lines are shown in color to make them easier to see.

4.4.2 [10] <§4.3> Consider a datapath similar to the one in Figure 4.11, but for a processor that only has one type of instruction: unconditional PC-relative branch. What would the cycle time be for this datapath?

unconditional branch의 Datapath는 왕의 그림과같다. Instruction memory → Sign-Extend → shift left 2 → Add → MUX

각 Component들의 latency를 모두더하면 → 200ps + 15ps + 10ps + 90ps + 200ps = 315ps

total latency = clock cycle time ∴ 315ps

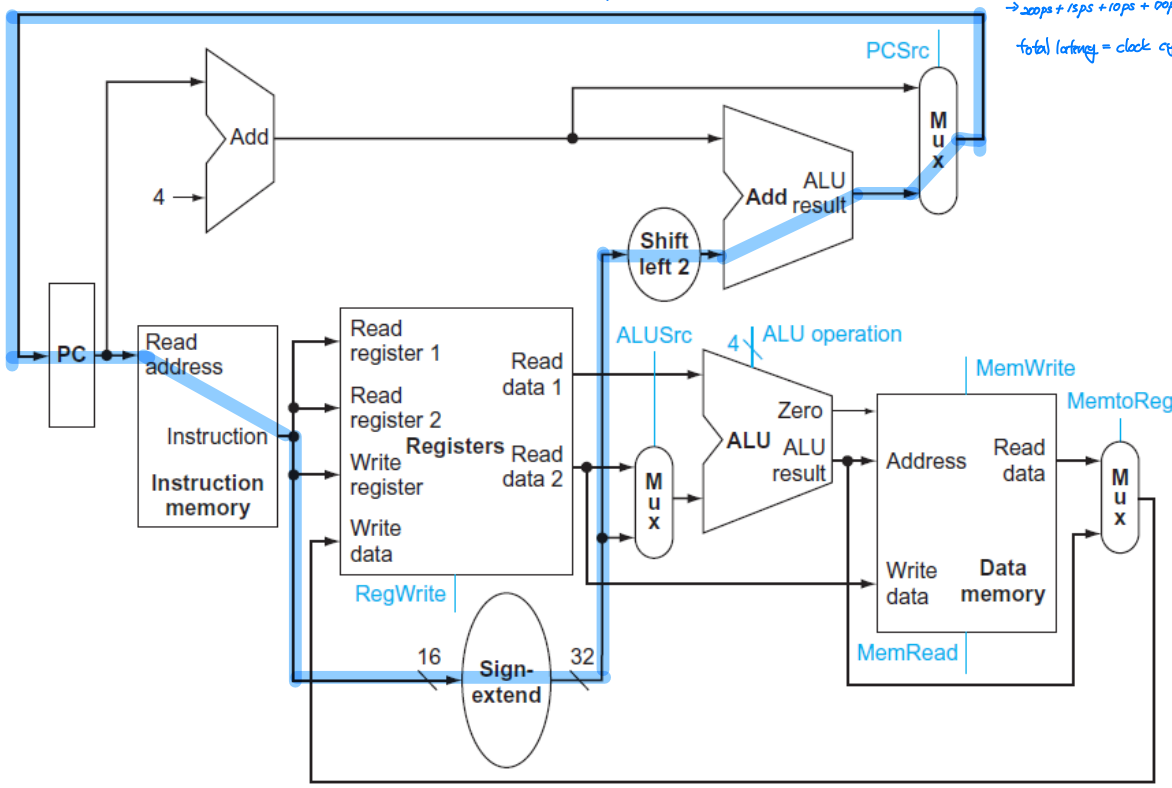

FIGURE 4.11 The simple datapath for the core MIPS architecture combines the elements required by different instruction classes. The components come from Figures 4.6, 4.9, and 4.10. This datapath can execute the basic instructions (load-store word, ALU operations, and branches) in a single clock cycle. Just one additional multiplexor is needed to integrate branches. The support for jumps will be added later.

stuck at zero fault와 stuck at 1 fault는 Logic의 결과가 항상 0, 1인것을 말한다. 이것을 테스트하기 위해선 Logic으로 들어갈 신호가 어떤 값이 되는지 봐야하는데 이것을 단 한번의 테스트로 확인하려면 어떤 signal을 보내도 0이 되어야하고 동시에 1이 되어야한다는 불가능하다.

4.6.2 [10] <§§4.3, 4.4> Repeat 4.6.1 for a stuck-at-1 fault. Can you use a single test for both stuck-at-0 and stuck-at-1? If yes, explain how; if no, explain why not.

(4.6.2 풀이 참고를 위해 첨부하였습니다. 4.6.1은 풀지 않으셔도 됩니다.)

4.6.1 [10] <§§4.3, 4.4> Let us assume that processor testing is done by filling the PC, registers, and data and instruction memories with some values (you can choose which values), letting a single instruction execute, then reading the PC, memories, and registers. These values are then examined to determine if a particular fault is present. Can you design a test (values for PC, memories, and registers) that would determine if there is a stuck-at-0 fault on this signal?

4.7 In this exercise we examine in detail how an instruction is executed in a single-cycle datapath. Problems in this exercise refer to a clock cycle in which the processor fetches the following instruction word:

10101100011000100000000000010100.

Assume that data memory is all zeros and that the processor's registers have the following values at the beginning of the cycle in which the above instruction word is fetched:

r0	r1	r2	r3	r4	r5	r6	r8	r12	r31
0	–1	2	–3	–4	10	6	8	2	–16

4.7.1 [5] <§4.4> What are the outputs of the sign-extend and the jump "Shift left 2" unit (near the top of Figure 4.24) for this instruction word?

Instruction [15:0] 를 sign-extend해줌 ⇒

0000 0000 0000 0000 0000 0000 0001 0100

Instruction [25:0]을 shift left 2 ⇒

0001 1000 1000 0000 0000 0101 0000

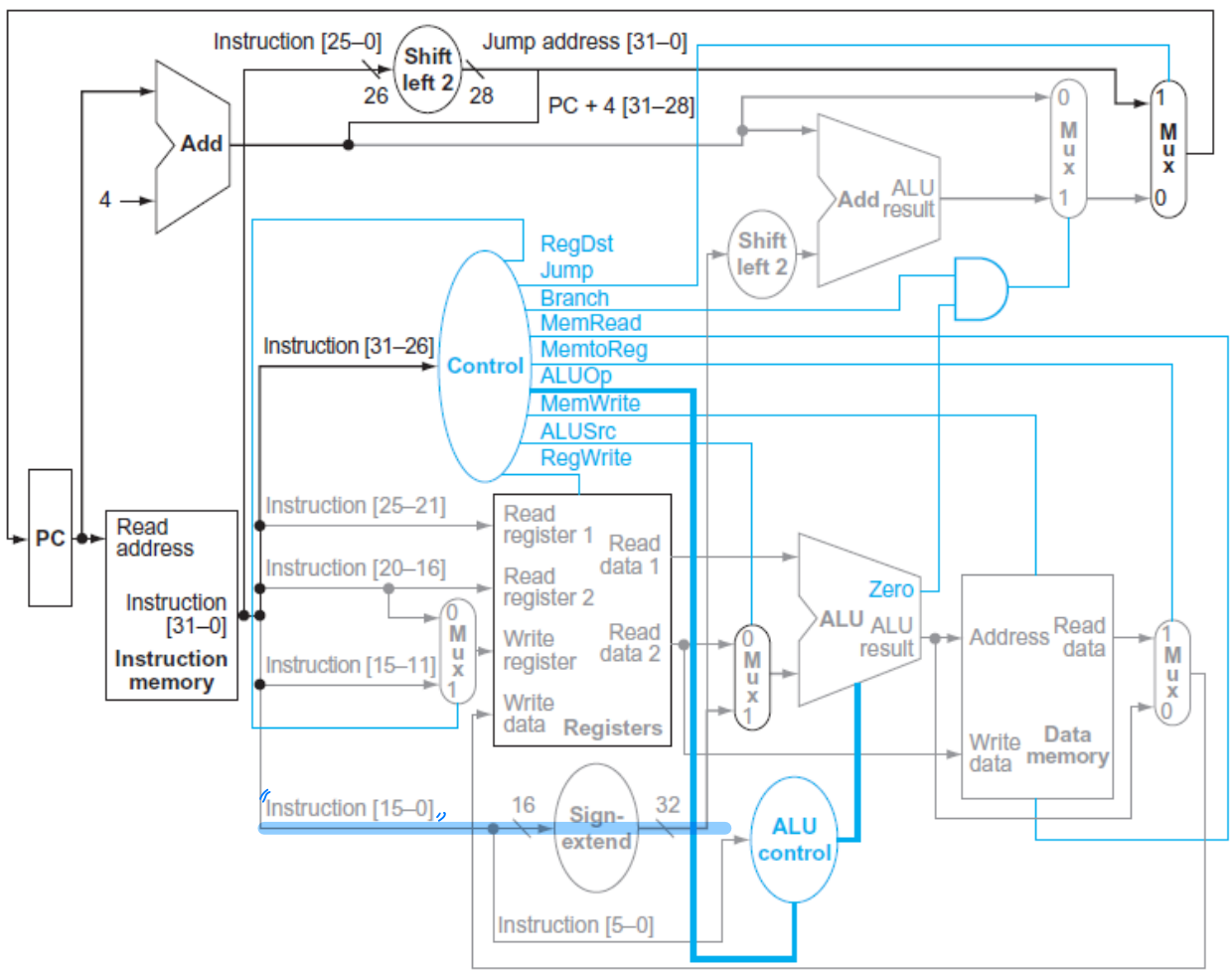

FIGURE 4.24 The simple control and datapath are extended to handle the jump instruction. An additional multiplexor (at the upper right) is used to choose between the jump target and either the branch target or the sequential instruction following this one. This multiplexor is controlled by the jump control signal. The jump target address is obtained by shifting the lower 26 bits of the jump instruction left 2 bits, effectively adding 00 as the low-order bits, and then concatenating the upper 4 bits of PC + 4 as the high-order bits, thus yielding a 32-bit address.

4.8 In this exercise, we examine how pipelining affects the clock cycle time of the processor. Problems in this exercise assume that individual stages of the datapath have the following latencies:

IF	ID	EX	MEM	WB
250ps	350ps	150ps	300ps	200ps

Also, assume that instructions executed by the processor are broken down as follows:

alu	beq	lw	sw
45%	20%	20%	15%

Instruction Decode를 split 한다.
why: 가장 긴 latency를 가지고 있음.

pipeline 에서는 가장 큰 latency가 clock cycle time 이다.
ID가 split되어 ID_1, ID_2가 각각 175ps를 가지므로
가장 큰 latency 이자 clock cycle time은 MEM 300ps

4.8.3 [10] <§4.5> If we can split one stage of the pipelined datapath into two new stages, each with half the latency of the original stage, which stage would you split and what is the new clock cycle time of the processor?

1.2 [5] <§1.2> The eight great ideas in computer architecture are similar to ideas from other fields. Match the eight ideas from computer architecture, "Design for Moore's Law", "Use Abstraction to Simplify Design", "Make the Common Case Fast", "Performance via Parallelism", "Performance via Pipelining", "Performance via Prediction", "Hierarchy of Memories", and "Dependability via Redundancy" to the following ideas from other fields: 빌딩의 엘레베이터는 문을열고,문닫고,운전기고. 내려가는등의 매뉴 일반적이고 반복되는, 단순한일을한다
Make the common case fast: rare case를 최적화하는것보다 일반적인 경우의 performance를 올리는 경향이 있기때문에 빌딩의 엘리베이터의 쓰임 에 강점이 있다.

a. Assembly lines in automobile manufacturing

b. Suspension bridge cables

c. Aircraft and marine navigation systems that incorporate wind information

d. Express elevators in buildings (d번만 풀이하시면 됩니다.)

Make the common case fast.

1.5 [4] <§1.6> Consider three different processors P1, P2, and P3 executing the same instruction set. P1 has a 3 GHz clock rate and a CPI of 1.5. P2 has a 2.5 GHz clock rate and a CPI of 2.2.

Clock → P1: $3GHz = 3\times10^9 Hz \times 10 = 30\times10^9/s$
P2: $2.5GHz = 2.5\times10^9 Hz \times 10 = 25\times10^9/s$
P3: $4.0GHz = 4.0\times10^9 Hz \times 10 = 40\times10^9/s$

Inst. → P1. $\frac{10}{15}\times30\times10^9 = 200\times10^9$
P2. $\frac{10}{22}\times25\times10^9 = 25\times10^9$
P3. $\frac{10}{22}\times40\times10^9 = \frac{200}{11}\times10^9$

a. Which processor has the highest performance expressed in instructions per second?

b. If the processors each execute a program in 10 seconds, find the number of cycles and the number of instructions.

Clock rate = Clock per Second = $\frac{\text{Clock}}{\text{Second}}$ → Clock rate × second = clock

$CPI = \frac{\text{Cycle}}{\text{Inst}}$ → $\frac{1}{CPI}\times\text{Cycle} = \text{Inst}$

c. We are trying to reduce the execution time by 30% but this leads to an increase of 20% in the CPI. What clock rate should we have to get this time reduction?

1.6 [20] <§1.6> Consider two different implementations of the same instruction set architecture. The instructions can be divided into four classes according to their CPI (class A, B, C, and D). P1 with a clock rate of 2.5 GHz and CPIs of 1, 2, 3, and 3, and P2 with a clock rate of 3 GHz and CPIs of 2, 2, 2, and 2.

Given a program with a dynamic instruction count of 1.0E6 instructions divided into classes as follows: 10% class A, 20% class B, 50% class C, and 20% class D, which implementation is faster? Total time: Program count (same program) × CPI × clock cycle time ⇒ CPI × $\frac{1}{\text{Clock rate}}$ = Total time

a. What is the global CPI for each implementation?

CPU Time = Instruction Count × CPI × Clock Cycle Time ⇒ CPI = CPU Time × Clock rate × $\frac{1}{\text{Instruction Count}}$

$\therefore$ CPI $P_1 = (10.4\times10^{-4}) \times (2.5\times10^{9}) \times \frac{1}{10^{6}} = 26\times10^{-1} = 2.6$

CPI $P_2 = (6.66\times10^{-4}) \times (3\times10^{9}) \times \frac{1}{10^{6}} = \underbrace{20.0}_{\text{올림자리에서 반올림}}\times10^{-1} = 2.0$

T_{P_1} : (Class 각각의 CPI) × $\frac{1}{\text{clock rate}}$

$= (1\times1\times10^{5} + 2\times2\times10^{5} + 5\times3\times10^{5} + 2\times3\times10^{5}) \times \frac{10}{25\times10^{9}}$

$= 10.4\times10^{-4}$s

T_{P_2} : $(1\times2\times10^{5} + 2\times2\times10^{5} + 5\times2\times10^{5} + 2\times2\times10^{5}) \times \frac{10}{30\times10^{9}}$

$= \underbrace{6.66\times10^{-4}}_{\text{3의 자리에서 반올림}}$s

$\therefore T_{P_1} > T_{P_2}$ T_{P_2}가 더 빠르다.

1.8 The Pentium 4 Prescott processor, released in 2004, had a clock rate of 3.6 GHz and voltage of 1.25 V. Assume that, on average, it consumed 10 W of static power and 90 W of dynamic power.

The Core i5 Ivy Bridge, released in 2012, had a clock rate of 3.4 GHz and voltage of 0.9 V. Assume that, on average, it consumed 30 W of static power and 40 W of dynamic power. 프로그램이 실행되면서 Device가 on 되어야하므로 Dynamic Power 사용.

Dynamic power $\propto \frac{1}{2} \times$ Capacitive loads $\times V^2 \times$ Frequency switched (clock rate의 개념)

1.8.1 [5] <§1.7> For each processor find the average capacitive loads.

Capacitive loads

⇒ 2 × Dynamic Power × $\frac{1}{V^2}$ × $\frac{1}{\text{clock rate}}$

단위 : F (패럿)

Pentium 4 : $2\times90\text{W}\times\left(\frac{100}{125}\right)^2\times\frac{10}{3.6\times10^{9}} \fallingdotseq 3.2\times10^{-8}$ F

Core I5 : $2\times40\text{W}\times\left(\frac{10}{9}\right)^2\times\frac{10}{3.4\times10^{9}} \fallingdotseq 29.0486\cdots\times10^{-9} \Rightarrow 2.9\times10^{-8}$ F

1.10 Assume a 15 cm diameter wafer has a cost of 12, contains 84 dies, and has 0.020 defects/cm². Assume a 20 cm diameter wafer has a cost of 15, contains 100 dies, and has 0.031 defects/cm². die area = wafer area / dies per wafer (wafer 반지름² × π / wafer area / Die area)

yield = $\frac{1}{(1 + (\text{Defects per area} \times \text{die area}/2))^2}$

1.10.3 [5] <§1.5> If the number of dies per wafer is increased by 10% and the defects per area unit increases by 15%, find the die area and yield.

die area (15cm : $(7.5)^2\times\pi\times\frac{1}{84}\times\frac{10}{11} = 1.9115\cdots = 1.91\text{cm}^2$

(단, 같은 조건에서 20cm : $(10)^2\times\pi\times\frac{1}{100}\times\frac{10}{11} = 2.854\cdots = 2.85\text{cm}^2$

$\text{yield}_{15} = 1/(1+(0.020\times1.15\times1.91\times\frac{1}{2}))^2 = 0.9504476\cdots$

$\text{yield}_{20} = 1/(1+(0.03\times1.15\times2.86\times\frac{1}{2}))^2 = 0.9081191\cdots$

1.11 The results of the SPEC CPU2006 bzip2 benchmark running on an AMD Barcelona has an instruction count of 2.389E12, an execution time of 750 s, and a reference time of 9650 s. CPU time = Instruction count × CPI × clock cycle time

여기서 Instruction count를 10% 증가시키면 CPI가 변화하고 Instruction Count는 clock cycle time이 영향을 주지않는다. CPU Time의 증가량은 구해지므로 Instruct count의 증가량과 같다.

1.11.3 [5] <§§1.6, 1.9> Find the increase in CPU time if the number of instructions of the benchmark is increased by 10% without affecting the CPI. 10 %

1.11.5 [5] <§§1.6, 1.9> Find the change in the SPECratio for this change.

SPEC ratio = reference time / CPU time

SPEC ratio의 변화량 = $\frac{\text{SPEC ratio (CPU time 증가후)}}{\text{SPEC ratio (CPU time 증가전)}}$

$= \frac{\text{reference time}/\text{CPU time}_{후}}{\text{reference time}/\text{CPU time}_{전}} = \frac{\text{CPU time}_{전}}{\text{CPU time}_{후}}$

1.11.3에서 CPU time이 10% 증가했으므로

SPEC ratio × $\frac{1}{10}$ ⇒ 10% 감소한다.

1.12 Section 1.10 cites as a pitfall the utilization of a subset of the performance equation as a performance metric. To illustrate this, consider the following two processors. P1 has a clock rate of 4 GHz, average CPI of 0.9, and requires the execution of 5.0E9 instructions. P2 has a clock rate of 3 GHz, an average CPI of 0.75, and requires the execution of 1.0E9 instructions.

1.12.1 [5] <§§1.6, 1.10> One usual fallacy is to consider the computer with the largest clock rate as having the largest performance. Check if this is true for P1 and P2. $n = \frac{P_1}{P_2} = \frac{T_2}{T_1}$

$T_{P_1} = (5\times10^9) \times 0.9 \times \frac{1}{4\times10^9} = 1.125s$

$T_{P_2} = (1\times10^9) \times 0.75 \times \frac{1}{3\times10^9} = 0.25s$

clock rate가 P_1이 더 크지만 P_1의 실행시간이 더 길다.

여기서 n은 P_1이 P_2보다 n배빠르다는뜻.

T = Instruction count × CPI × $\frac{1}{\text{Clock cycle time}}$

T 실행시간

∴ clock rate과 Execution Time은 비례관계가 아니다.

1.13 Another pitfall cited in Section 1.10 is expecting to improve the overall performance of a computer by improving only one aspect of the computer. Consider a computer running a program that requires 250 s, with 70 s spent executing FP instructions, 85 s executed L/S instructions, and 40 s spent executing branch instructions.

1.13.2 [5] <§1.10> By how much is the time for INT operations reduced if the total time is reduced by 20%?

before reduced : $T_{Total}=250$, $INT = T_{Total} - (FP + LS + branch)$
$= 250 - 195 = 55$

After reduced : $T_{total} = 250 \times 0.8 = 200$, $INT = T_{Total} - (FP+LS+branch)$
$= 200 - 195 = 5$

Reduction time INT : 91 %

1.14 Assume a program requires the execution of 50 × 106 FP instructions, 110 × 106 INT instructions, 80 × 106 L/S instructions, and 16 × 106 branch instructions. The CPI for each type of instruction is 1, 1, 4, and 2, respectively. Assume that the processor has a 2 GHz clock rate.

1.14.1 [10] <§1.10> By how much must we improve the CPI of FP instructions if we want the program to run two times faster?

Clock Cycle = CPI × Number of Inst

$\Rightarrow (1\times50) + (1\times110) + (4\times80) + (2\times16)$

$\Rightarrow 512\times10^6$

Clock rate = 2×10^9 Hz

$T_{Total} = \frac{512\times10^6}{2\times10^9} = 0.256s$

Clock cycle을 절반으로줄이면 프로그램을 2배 빠르게할 수 있다.

$CPI_{FP}\times50 + (1\times110) + (4\times80) + (2\times16) = \frac{\text{Clock cycle}}{2} = 256$

$CPI_{FP}\times50 = 256 - 462$

CPI_{FP}가 음수가 되어야하므로 프로그램을 2배 빠르게 하는건 불가능하다.