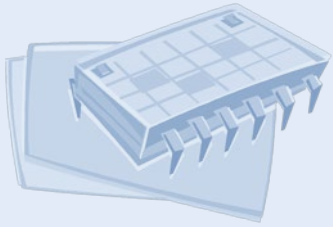


Multicore System Design @ Hallym University

이정근

대세는 멀티코어~


AMD
Phenom II
X4


intel
CORE i7
inside

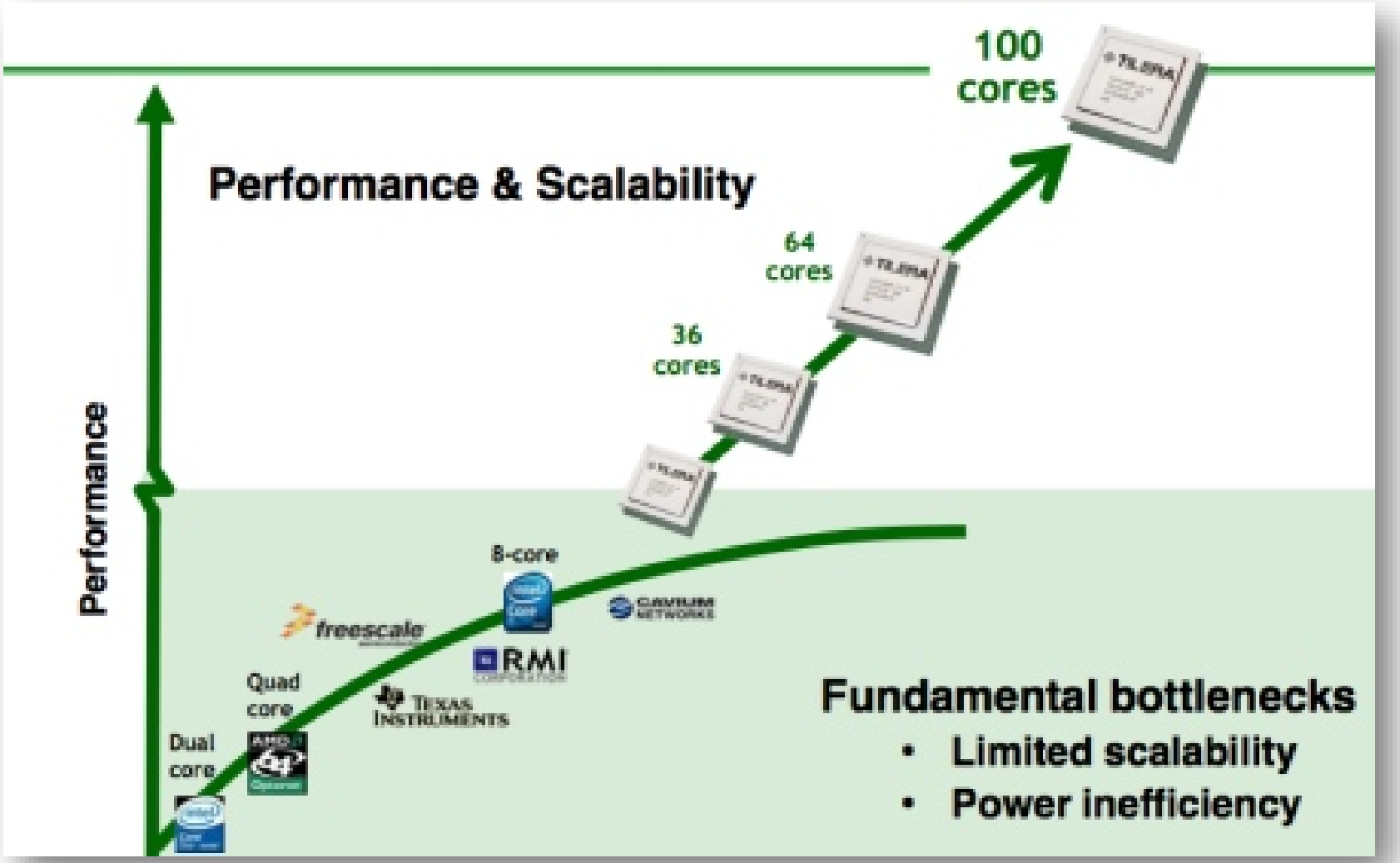

© Original Artist
Reproduction rights obtainable from
www.CartoonStock.com
MULTI CORE
MULTI CORE
AMD
intel
MULTI CORE MOUNTAIN

소개~

- “Multi-Core” 프로세서란 ?
  - A **multi-core** CPU (or chip-level multiprocessor, CMP) combines two or more independent cores into a single package composed of a single integrated circuit (IC), called a die, or more dies packaged together.

    - Wikipedia

Recent News: Challenging X86



Multicore @ Hallym University

- 단일 Core 구조: 간략화된 MIPS 프로세서

새 기능 로그인 / 계정 만들기

문서 토론 읽기 편집 역사 검색

위키백과
우리 모두의 백과사전

더 사용하기 편리해진 위키백과, 조금 바뀐 것이 느껴지시나요? 자세히 알아 보기 [숨기기] [번역에 도움이 필요합니다!]

대문
사용자 모임
요즘 화제
최근 바뀜
모든 문서 보기
임의 문서로
도움말
기부 안내

도구모음
여기를 가리키는 문서
가리키는 글의 바뀜
특수 문서 목록
인쇄용 문서
고유링크
이 문서 인용하기

다른 언어
Česky
Чăвашла
Deutsch
English
Español
Suomi
Français
Italiano
日本語

MIPS 아키텍처

위키백과 — 우리 모두의 백과사전.

MIPS(**M**icroprocessor without **I**nterlocked **P**ipeline **S**tages)는 밉스 테크놀로지에서 개발한 RISC 마이크로프로세서이다.

MIPS 디자인은 실리콘 그래픽스 사의 컴퓨터 시스템, 많은 임베디드 시스템과 윈도 CE 장치, 시스코 시스템즈 라우터에 사용되었다. 그 외에도 소니 플레이스테이션, 플레이스테이션 2, 플레이스테이션 포터블 같은 게임 콘솔에도 사용되었다. 한편 국내에서는 아이스테이션 사의 아이스테이션 T43제품에 쓰인 바 있다.

도시바에서 생산한 MIPS R4400 프로세서.

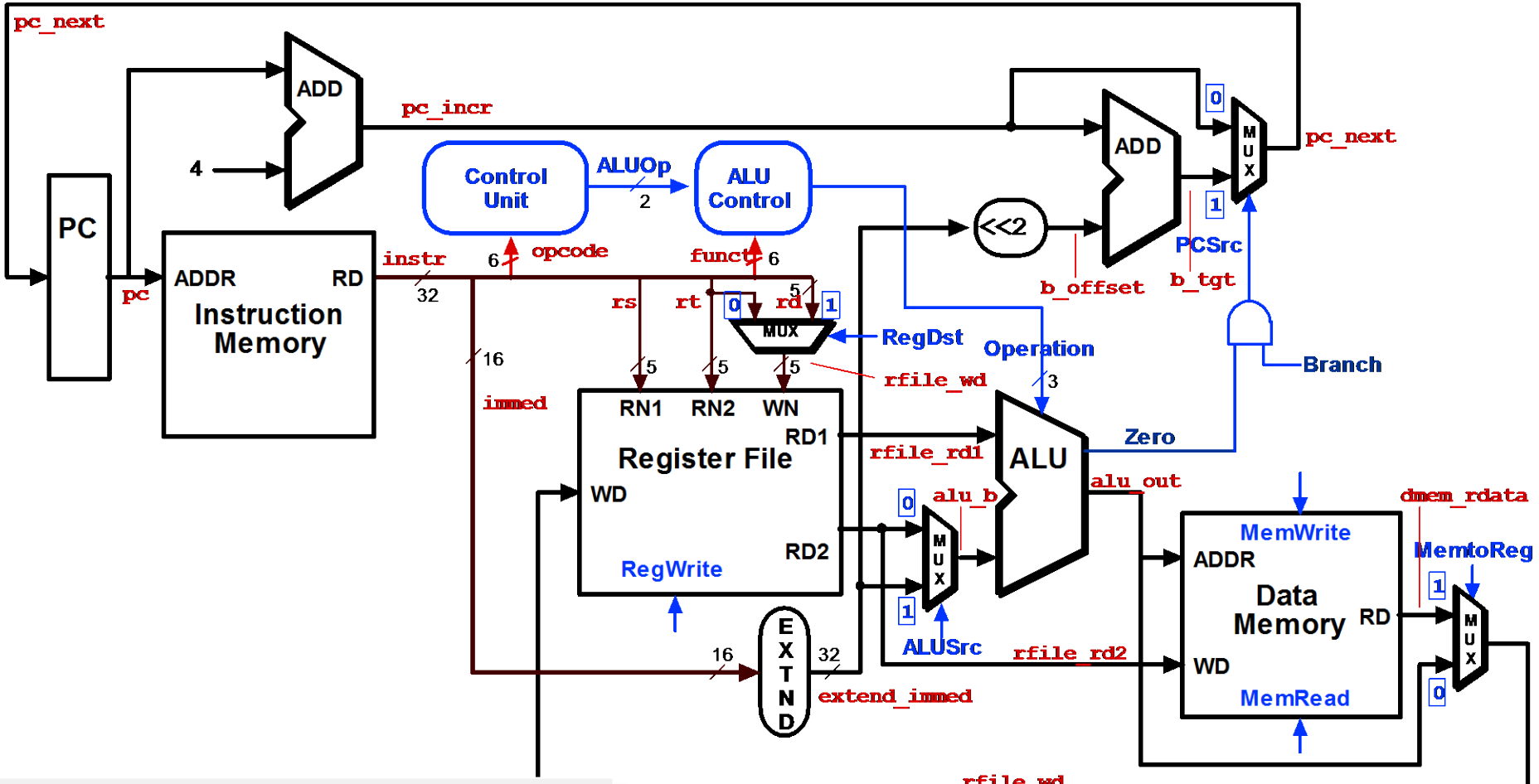
초기 MIPS 아키텍처는 32비트를 사용하였다. 이들은 32비트 레지스터와 데이터 경로를 가지고 있었다. 그 이후의 MIPS 프로세서들은 64비트 구현을 사용하였다. 5종류의 하위 호환성 MIPS 동작 세트가 존재하며, 이들은 각각 MIPS I, MIPS II, MIPS III, MIPS IV, MIPS 32/64라고 불린다. MIPS 32/64 릴리즈 2에서는 동작 세트와 함께 컨트롤 레지스터 셋도 정의하고 있다. MIPS-3D 같은 3차원 그래픽을 위한 SIMD 확장 기능도 존재한다. MDMX (MaDMaX) 확장은 64비트 유동 소수점 레지스터를 활용하는 정수 연산 집합이다. 최근에는 MIPS MT라고 하는 인텔 펜티엄 4 프로세서의 하이퍼스레딩 같은 멀티스레딩 기능이 추가되었다.

MIPS 디자이너들이 명령어 세트를 깔끔하게 설계했기 때문에 많은 대학의 컴퓨터 아키텍처 강좌에서 MIPS 아키텍처를 가르친다. MIPS 프로세서의 디자인은 후기 RISC 아키텍처에 큰 영향을 주었다.

특징

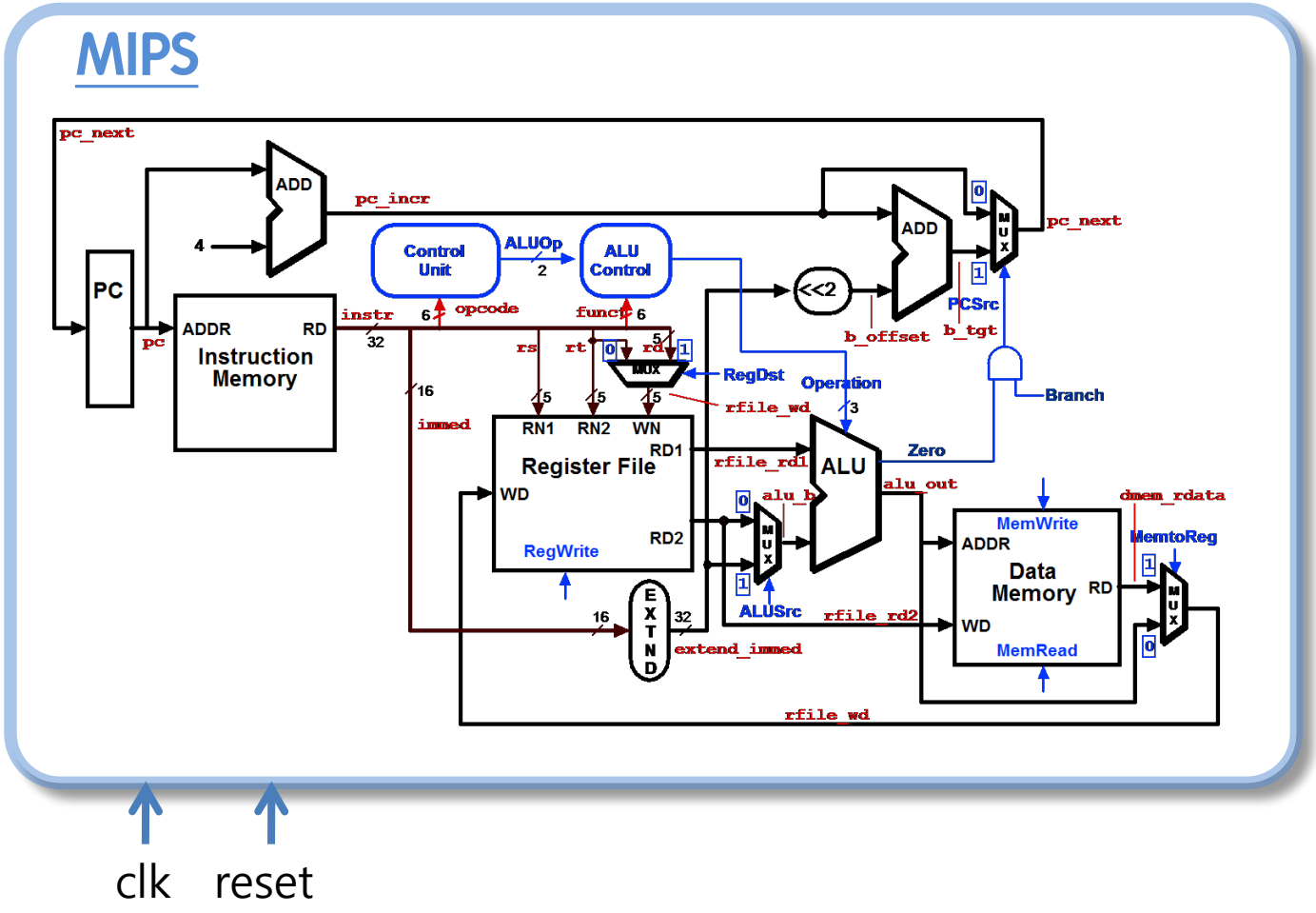
- 32 비트 머신
- 32 레지스터
- **명령어**
  - **ALU 연산**
    - add/sub
    - and/or
    - slt
    - sll / srl
  - **MEMORY 접근**
    - lw (load word)
    - sw (store word)
  - **BRANCH**
    - beq

Simple MIPS 프로세서 구조도



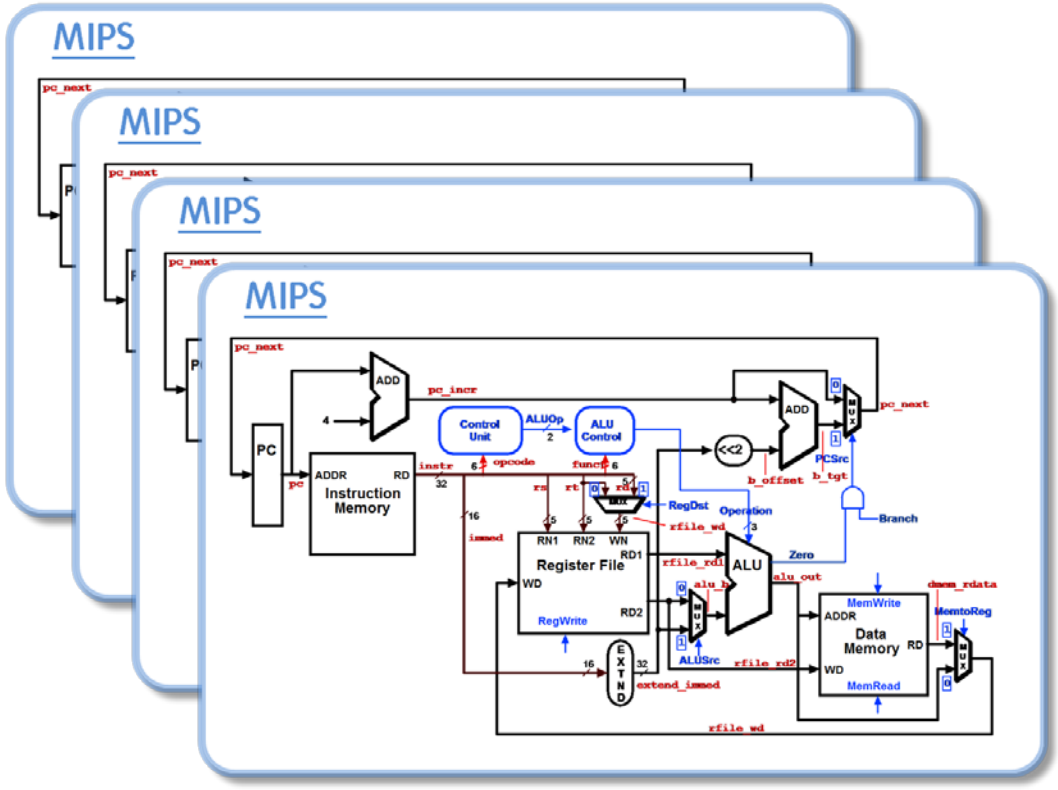
Input							Output								
OP	Op5	Op4	Op3	Op2	Op1	Op0	RegDst	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	ALUOp1	ALUOp0
RT	0	0	0	0	0	0	1	0	0	1	0	0	0	1	0
lw	1	0	0	0	1	1	0	1	1	1	1	0	0	0	0
sw	1	0	1	0	1	1	X	1	X	0	0	1	0	0	0
beq	0	0	0	1	0	0	X	0	X	0	0	0	1	0	1

Simple MIPS 프로세서 구조도

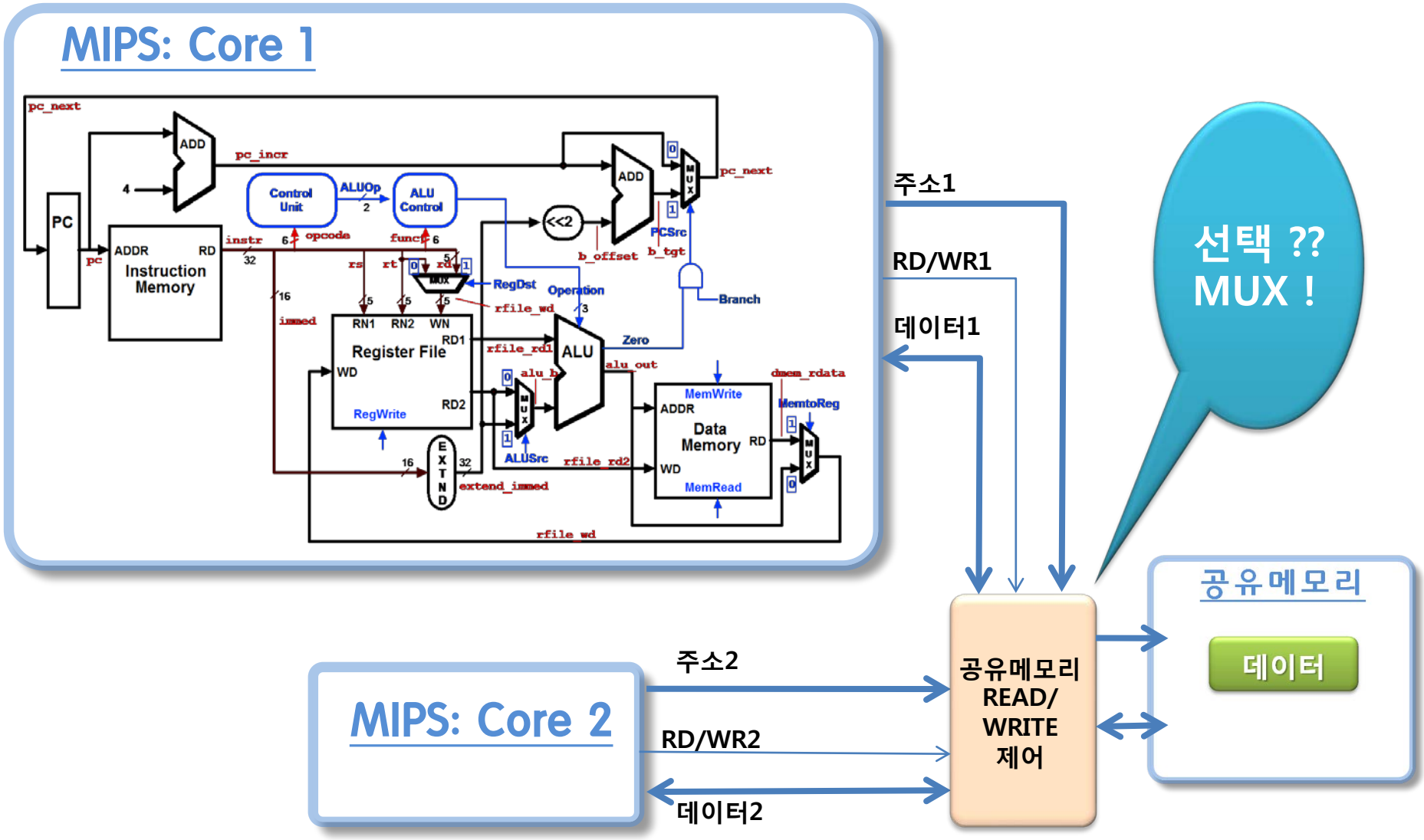


멀티코어 확장

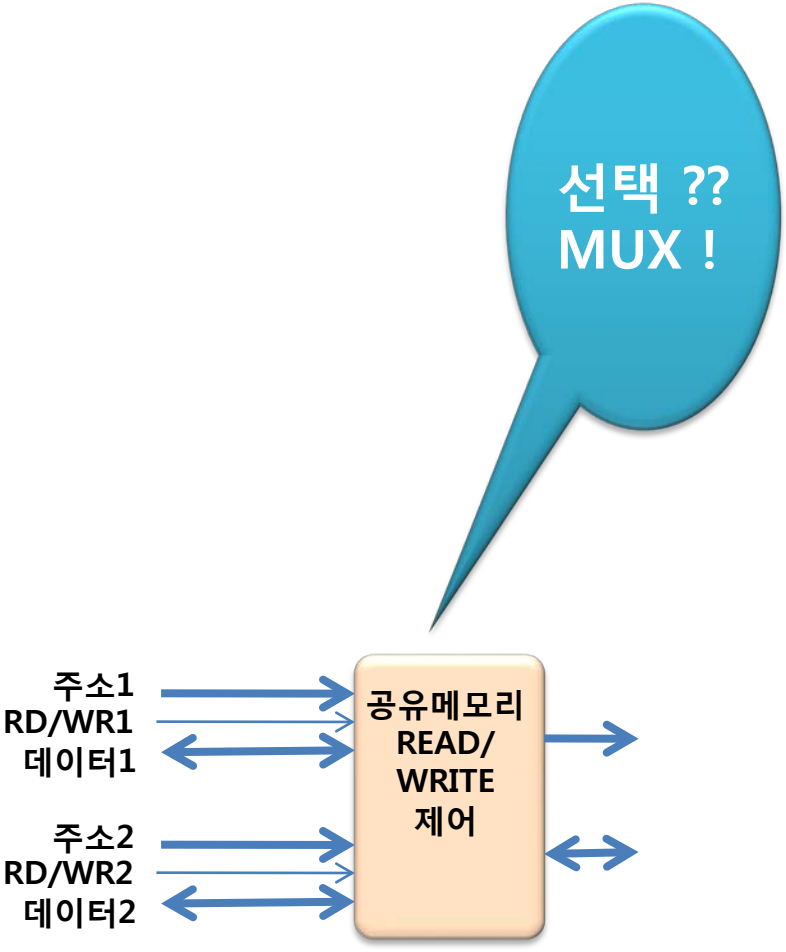
- 멀티코어간 통신 메커니즘
  - 공유 메모리 (Shared Memory)
  - 메시지 패싱 (Message Passing): On-Chip Network



Dual-Core 구성도



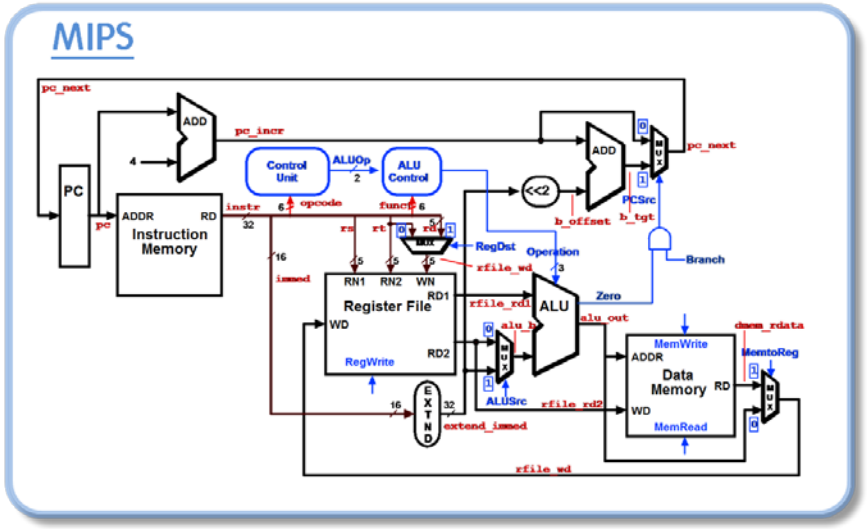
Dual-Core 구성도



각각의 Core에서 가져온 공유메모리에 대한 접근 정보를 이용하여 MUX의 선택 신호를 만들어냄

각각의 Core에서 가져온 공유메모리에 대한 접근 정보 ?

각 코어의 공유메모리 접근 정보



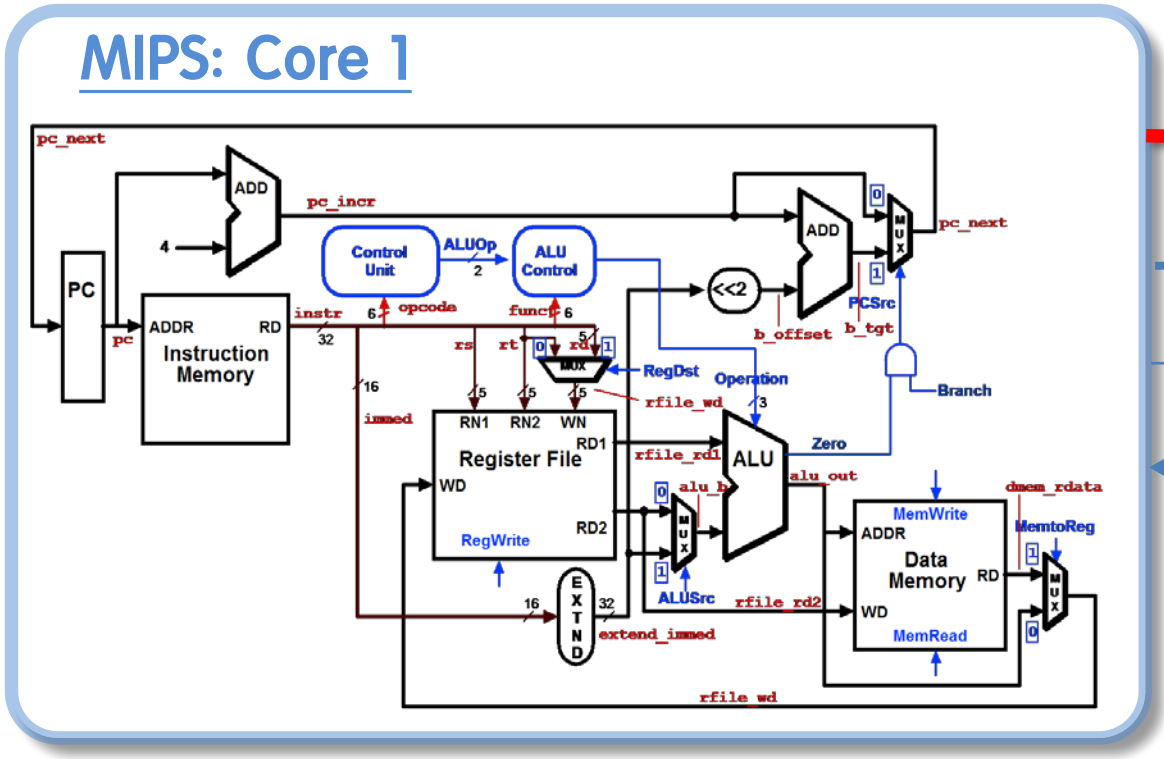
공유메모리

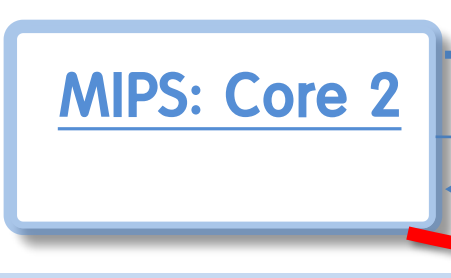
데이터

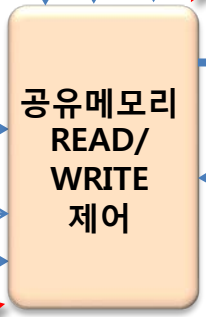
- 각 프로세서의 데이터 메모리 주소 체계
  * 0 ~ X-1 번지 : 내부 메모리
  * X 번지 ~ : 공유메모리

→ Address 분석 후 공유메모리 주소에 해당하면 **sharedMem** 신호 생성

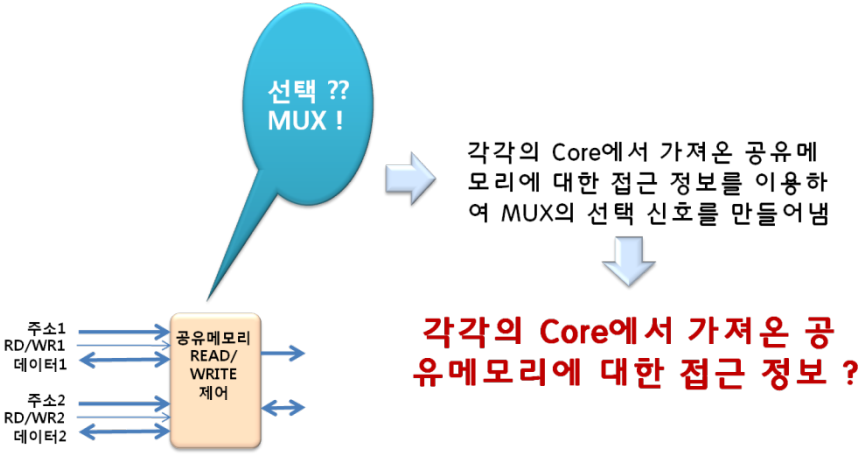
각 코어의 공유메모리 접근 정보

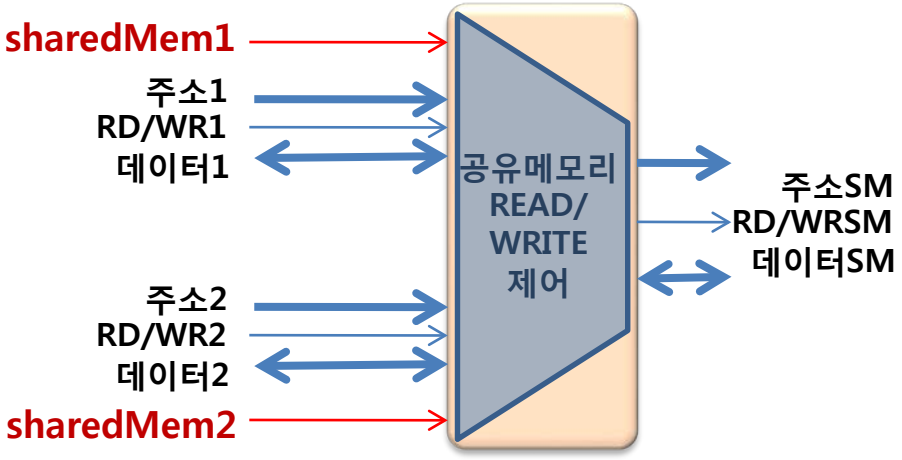






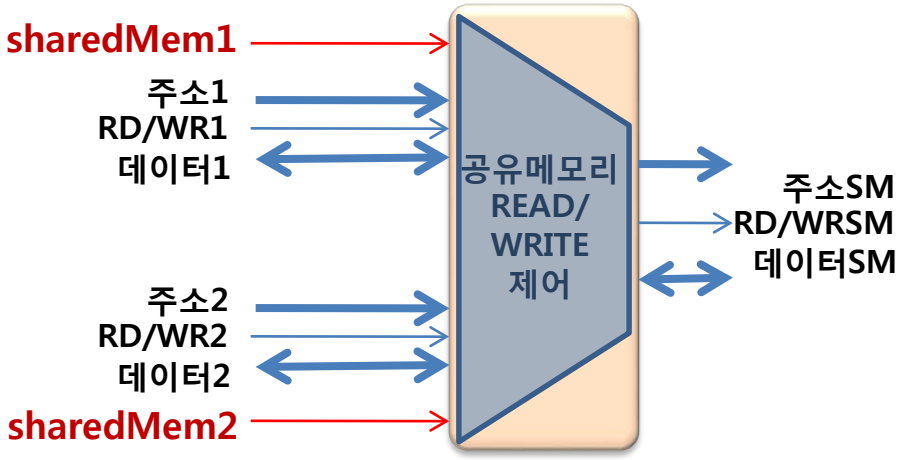
공유메모리 READ/WRITE 제어





sharedMem1	sharedMem2	
0	0	내부접근
0	1	Core2가 접근
1	0	Core1이 접근
1	1	허용안됨

공유메모리 READ/WRITE 제어

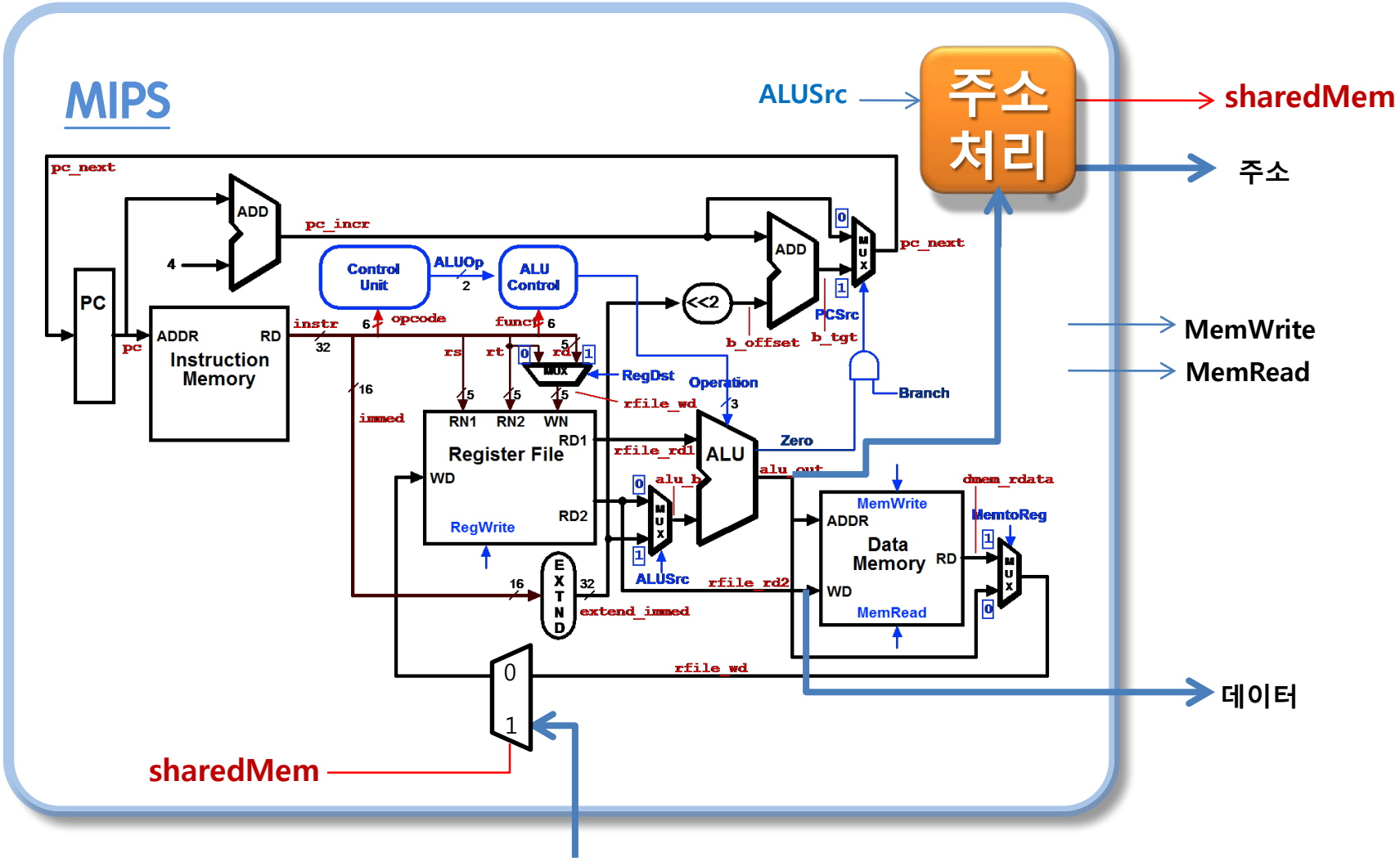


sharedMem1	sharedMem2	
0	0	Core2가 접근
0	1	Core2가 접근
1	0	Core1이 접근
1	1	허용안됨

Default Core의 경우
임의로 읽힌 데이터가 내부에서 받아들여지지 않도록 Core 내부의 MUX로 제어함!

K개의 core들을 갖는 Manycore 시스템 → K-to-1 MUX를 가지고 공유메모리 제어

Core 내부의 처리 모듈

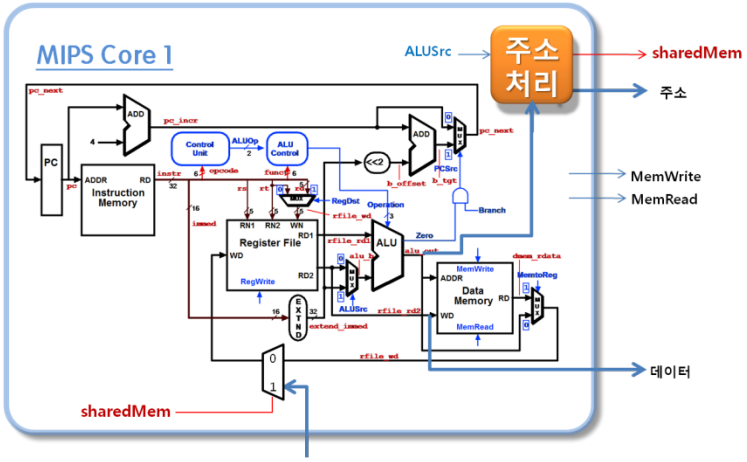


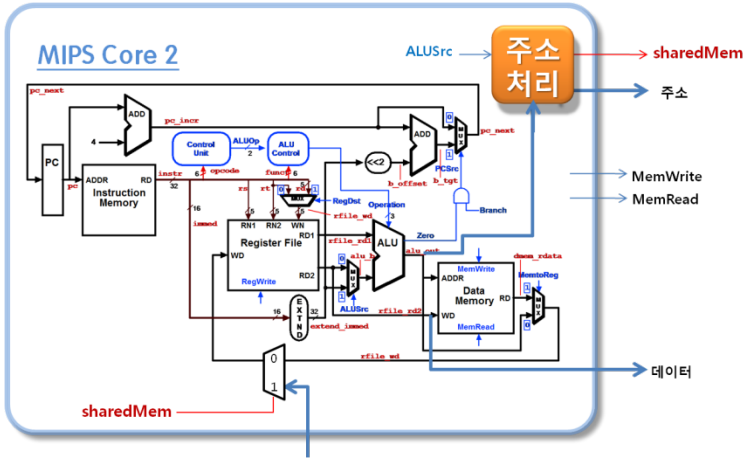
소프트웨어 on 멀티코어~

- 1 ~ 1000 더하기
  - ResultCore1 ← 1 ~ 500 더하기 @ Core 1
  - ResultCore2 ← 501 ~ 1000 더하기 @ Core 2
  - TotalResult ← ResultCore1+ResultCore2 @ Core1
  - 4~8개의 Core를 포함한 시스템으로 확장

- 정렬: 많은 주요 SW의 핵심 자료처리 엔진
  - 1000 아이템 정렬
  - 1 ~ 500 정렬 @ Core 1
  - 501 ~ 1000 정렬 @ Core 2
  - Merge @ Core 1

1 ~ 1000 더하기@DualCore





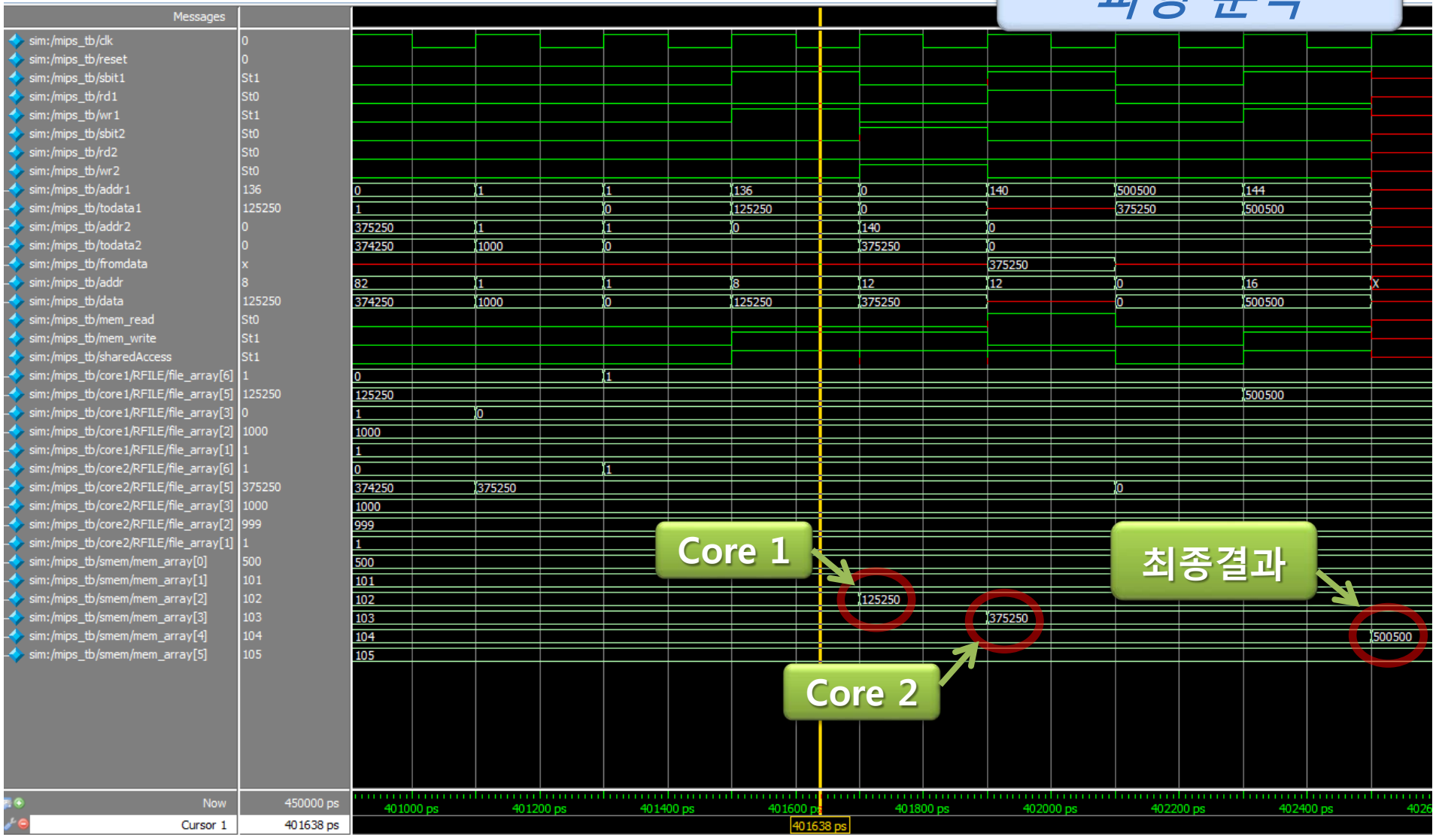

```
5'd0 : data_out = { 6'd35, 5'd0, 5'd1, 16'd0 };              // lw $1, 0($0)    r1=1
5'd1 : data_out = { 6'd35, 5'd0, 5'd2, 16'd4 };              // lw $2, 4($0)    r2=1000
5'd2 : data_out = { 6'd0,  5'd0, 5'd2, 5'd3, 5'd1, 6'd2};    // srl r3 r2 1  :: r3 <- 500
5'd3 : data_out = { 6'd43, 5'd0, 5'd3, 16'd128 };            // SharedMEM[0] <- MEM[32] = $3
5'd4 : data_out = { 6'd0,  5'd0, 5'd0, 5'd5, 5'd0, 6'd32 };   // add $5, $0, $0  r5 = 0 : NOP
//
5'd5 : data_out = { 6'd0,  5'd0, 5'd0, 5'd5, 5'd0, 6'd32 };   // add $5, $0, $0  r5 = 0
5'd6 : data_out = { 6'd0,  5'd3, 5'd5, 5'd5, 5'd0, 6'd32 };   // add $5, $5, $3  r5 = r5 + r3
5'd7 : data_out = { 6'd0,  5'd3, 5'd1, 5'd3, 5'd0, 6'd34 };   // add $3, $3, #1  r3 = r3 - r1
5'd8 : data_out = { 6'd0,  5'd3, 5'd1, 5'd6, 5'd0, 6'd42 };   // slt $6, $3, $1  if r3 < r1 ?
5'd9 : data_out = { 6'd4,  5'd6, 5'd0, -16'd4 };              // beq $6, $zero, -4  if not, go back 3
//
5'd10: data_out = { 6'd43, 5'd0, 5'd5, 16'd136 };            // Shared MEM[2] = $5 = 1+2+ ... + 50
5'd11: data_out = { 6'd0,  5'd0, 5'd0, 5'd9, 5'd0, 6'd32 };   // add $5, $0, $0  r9 = 0 : NOP
5'd12: data_out = { 6'd35, 5'd0, 5'd7, 16'd140 };            // lw $2, 140($0)   r7 <- Result of Core 2
5'd13: data_out = { 6'd0,  5'd5, 5'd7, 5'd5, 5'd0, 6'd32 };   // add $5, $5, $7  r5 = r5 + r7
5'd14: data_out = { 6'd43, 5'd0, 5'd5, 16'd144 };            // Shared MEM[2] = 1+2+ ... + 50 + ... + 100
```

```
5'd0 : data_out = { 6'd35, 5'd0, 5'd1, 16'd0 };              // lw $1, 0($0)    r1=1
5'd1 : data_out = { 6'd35, 5'd0, 5'd2, 16'd4 };              // lw $2, 4($0)    r2=100
5'd2 : data_out = { 6'd0,  5'd2, 5'd1, 5'd2, 5'd0, 6'd34 };   // add $2, $2, #1  r2 = r2 - r1
5'd3 : data_out = { 6'd0,  5'd0, 5'd0, 5'd5, 5'd0, 6'd32 };   // add $5, $0, $0  r5 = 0 : NOP
// Core 1 Store a Number to MEM[128]:Shared Memory
5'd4 : data_out = { 6'd35, 5'd0, 5'd3, 16'd128 };            // lw $2, 128($0)   r3 <- number : 500
//
5'd5 : data_out = { 6'd0,  5'd0, 5'd0, 5'd5, 5'd0, 6'd32 };   // add $5, $0, $0  r5 = 0
5'd6 : data_out = { 6'd0,  5'd3, 5'd1, 5'd3, 5'd0, 6'd32 };   // add $3, $3, #1  r3 = r3 + r1
5'd7 : data_out = { 6'd0,  5'd3, 5'd5, 5'd5, 5'd0, 6'd32 };   // add $5, $5, $3  r5 = r5 + r3
5'd8 : data_out = { 6'd0,  5'd2, 5'd3, 5'd6, 5'd0, 6'd42 };   // slt $6, $2, $3  if r2 < r3 ?
5'd9 : data_out = { 6'd4,  5'd6, 5'd0, -16'd4 };             // beq $6, $0, -4  if not, go back 3
//
5'd10: data_out = { 6'd0,  5'd0, 5'd0, 5'd9, 5'd0, 6'd32 };   // add $5, $0, $0  r9 = 0 : NOP
5'd11: data_out = { 6'd43, 5'd0, 5'd5, 16'd140 };            // Shared MEM[2] = $5 = 1+2+ ... + 50
5'd12: data_out = { 6'd0,  5'd0, 5'd0, 5'd5, 5'd0, 6'd32 };   // add $5, $0, $0  r5 = 0 : NOP
5'd13: data_out = { 6'd0,  5'd0, 5'd0, 5'd5, 5'd0, 6'd32 };   // add $5, $0, $0  r5 = 0 : NOP
5'd14: data_out = { 6'd0,  5'd0, 5'd0, 5'd5, 5'd0, 6'd32 };   // add $5, $0, $0  r5 = 0 : NOP
```

코어간 데이터 동기화를 위하여 NOP (No Operation) 사용 !

1 ~ 1000 더하기@DualCore

파형 분석



코어별 수행 코드

Core1

Core1 - 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

```
Core 1
        // Head Computation: 분할정보 계산하고 다른 코어로 정보를 전달하기 위해 공유메모리에 저장하기
        5'd0 : data_out = { 6'd35, 5'd0, 5'd1, 16'd0 };              // lw $1, 0($0)   r1=1
        5'd1 : data_out = { 6'd35, 5'd0, 5'd2, 16'd4 };              // lw $2, 4($0)   r2=100
        5'd2 : data_out = { 6'd0,  5'd0, 5'd2, 5'd3, 5'd1, 6'd2};    // srl r3 r2 "1"  나누기 $3 == 50
        5'd3 : data_out = { 6'd43, 5'd0, 5'd3, 16'd128 };            // SharedMEM[0] <- MEM[32] = $3
        5'd4 : data_out = { 6'd0,  5'd0, 5'd2, 5'd4, 5'd2, 6'd2};    // srl r4 r2 "2"  나누기 $4 == 25
        5'd5 : data_out = { 6'd43, 5'd0, 5'd4, 16'd132 };            // SharedMEM[1] <- MEM[33] = $4
        5'd6 : data_out = { 6'd0,  5'd3, 5'd4, 5'd5, 5'd0, 6'd32 };  // add $5, $5, $3  r5 = r4 + r3 == 75
        5'd7 : data_out = { 6'd43, 5'd0, 5'd5, 16'd136 };            // SharedMEM[2] <- MEM[34] = $5
        5'd8 : data_out = { 6'd0,  5'd0, 5'd0, 5'd30, 5'd0, 6'd32 }; // add $30, $0, $0  r30 = 0 : NOP
        5'd9 : data_out = { 6'd0,  5'd0, 5'd0, 5'd30, 5'd0, 6'd32 }; // r30 = 0 : NOP
        5'd10: data_out = { 6'd0,  5'd0, 5'd0, 5'd30, 5'd0, 6'd32 }; // r30 = 0 : NOP
        5'd11: data_out = { 6'd0,  5'd0, 5'd0, 5'd30, 5'd0, 6'd32 }; // r30 = 0 : NOP
        5'd12: data_out = { 6'd0,  5'd0, 5'd0, 5'd30, 5'd0, 6'd32 }; // r30 = 0 : NOP
        5'd13: data_out = { 6'd0,  5'd0, 5'd0, 5'd30, 5'd0, 6'd32 }; // r30 = 0 : NOP

        // Body Computation
        5'd14: data_out = { 6'd0,  5'd0, 5'd0, 5'd5, 5'd0, 6'd32 };  // add $5, $0, $0  r5 = 0
        5'd15: data_out = { 6'd0,  5'd4, 5'd5, 5'd5, 5'd0, 6'd32 };  // add $5, $5, $4  r5 = r5 + r4
        5'd16: data_out = { 6'd0,  5'd4, 5'd1, 5'd4, 5'd0, 6'd34 };  // add $3, $3, #1  r4 = r4 - r1
        5'd17: data_out = { 6'd0,  5'd4, 5'd1, 5'd6, 5'd0, 6'd42 };  // slt $6, $3, $1  if r4 < r1 ?
        5'd18: data_out = { 6'd4,  5'd6, 5'd0, -16'd4 };             // beq $6, $zero, -4  if not, go back 3

        // r5 <- 1+2+...+25
        // Tail Computation: 분산된 다수의 코어에 의해서 계산된 값을 모아 최종값을 계산
        5'd19: data_out = { 6'd43, 5'd0, 5'd5, 16'd140 };            // Shared MEM[3] = $5 = 1+2+ ... + 25

        // 각 프로세서들이 계산을 결과를 정장하는 시간을 기다린다: Synchronization
        5'd20: data_out = { 6'd0,  5'd0, 5'd0, 5'd30, 5'd0, 6'd32 }; // r30 = 0 : NOP Core2 SW
        5'd21: data_out = { 6'd0,  5'd0, 5'd0, 5'd30, 5'd0, 6'd32 }; // r30 = 0 : NOP Core3 SW
        5'd22: data_out = { 6'd0,  5'd0, 5'd0, 5'd30, 5'd0, 6'd32 }; // r30 = 0 : NOP Core4 SW

        // 다른 Core들의 계산 결과가 저장된 공유메모리로 부터 결과를 하나씩 읽어들임
        5'd23: data_out = { 6'd35, 5'd0, 5'd6, 16'd144 };            // lw $6, 144($0): r6 <- Result of Core 2
        5'd24: data_out = { 6'd35, 5'd0, 5'd7, 16'd148 };            // lw $7, 148($0): r7 <- Result of Core 3
        5'd25: data_out = { 6'd35, 5'd0, 5'd8, 16'd152 };            // lw $8, 152($0): r8 <- Result of Core 4

        //
        5'd26: data_out = { 6'd0,  5'd5, 5'd6, 5'd5, 5'd0, 6'd32 };  // add $5, $5, $6  r5 = r5 + r6
        5'd27: data_out = { 6'd0,  5'd5, 5'd7, 5'd5, 5'd0, 6'd32 };  // add $5, $5, $6  r5 = r5 + r7
        5'd28: data_out = { 6'd0,  5'd5, 5'd8, 5'd5, 5'd0, 6'd32 };  // add $5, $5, $6  r5 = r5 + r8
        5'd29: data_out = { 6'd43, 5'd0, 5'd5, 16'd128 };            // Shared MEM[0] = 1+...+100
```

Core2

```
        5'd0 : data_out = { 6'd35, 5'd0, 5'd1, 16'd0 };              // lw $1, 0($0)   r1=1
        // 데이터 동기화를 위한 NOPs
        5'd1 : data_out = { 6'd0,  5'd0, 5'd0, 5'd30, 5'd0, 6'd32 }; // r30 = 0 : NOP
        5'd2 : data_out = { 6'd0,  5'd0, 5'd0, 5'd30, 5'd0, 6'd32 }; // r30 = 0 : NOP
        5'd3 : data_out = { 6'd0,  5'd0, 5'd0, 5'd30, 5'd0, 6'd32 }; // r30 = 0 : NOP
        5'd4 : data_out = { 6'd35, 5'd0, 5'd2, 16'd128 };            // r2 <- SharedMEM[0] = 50
        5'd5 : data_out = { 6'd0,  5'd2, 5'd1, 5'd2, 5'd0, 6'd34 };  // add $2, $2, #1  r2 = r2 - r1
        5'd6 : data_out = { 6'd35, 5'd0, 5'd3, 16'd132 };            // r3 <- SharedMEM[1] = 25
        5'd7 : data_out = { 6'd0,  5'd2, 5'd1, 5'd2, 5'd0, 6'd34 };  // add $2, $2, #1  r2 = r2 - r1
        5'd8 : data_out = { 6'd0,  5'd0, 5'd0, 5'd30, 5'd0, 6'd32 }; // r30 = 0 : NOP
        5'd9 : data_out = { 6'd0,  5'd0, 5'd0, 5'd30, 5'd0, 6'd32 }; // r30 = 0 : NOP
        5'd10: data_out = { 6'd0,  5'd0, 5'd0, 5'd30, 5'd0, 6'd32 }; // r30 = 0 : NOP
        5'd11: data_out = { 6'd0,  5'd0, 5'd0, 5'd30, 5'd0, 6'd32 }; // r30 = 0 : NOP
        5'd12: data_out = { 6'd0,  5'd0, 5'd0, 5'd30, 5'd0, 6'd32 }; // r30 = 0 : NOP
        5'd13: data_out = { 6'd0,  5'd0, 5'd0, 5'd30, 5'd0, 6'd32 }; // r30 = 0 : NOP

        // 6 + 27 + ... + 50
        5'd14: data_out = { 6'd0,  5'd0, 5'd0, 5'd5, 5'd0, 6'd32 };  // add $5, $0, $0  r5 = 0
        5'd15: data_out = { 6'd0,  5'd3, 5'd1, 5'd3, 5'd0, 6'd32 };  // add $3, $3, #1  r3 = r3 + r1
        5'd16: data_out = { 6'd0,  5'd3, 5'd5, 5'd5, 5'd0, 6'd32 };  // add $5, $5, $3  r5 = r5 + r3
        5'd17: data_out = { 6'd0,  5'd2, 5'd3, 5'd6, 5'd0, 6'd42 };  // slt $6, $2, $3  if r2 < r3 ?
        5'd18: data_out = { 6'd4,  5'd6, 5'd0, -16'd4 };             // beq $6, $0, -4  if not, go back 3
        5'd19: data_out = { 6'd0,  5'd0, 5'd0, 5'd30, 5'd0, 6'd32 }; // r30 = 0 : NOP

        5'd20: data_out = { 6'd43, 5'd0, 5'd5, 16'd144 };            // 결과 저장
        5'd21: data_out = { 6'd0,  5'd0, 5'd0, 5'd30, 5'd0, 6'd32 }; // r30 = 0 : NOP
        5'd22: data_out = { 6'd0,  5'd0, 5'd0, 5'd30, 5'd0, 6'd32 }; // r30 = 0 : NOP
        5'd23: data_out = { 6'd0,  5'd0, 5'd0, 5'd30, 5'd0, 6'd32 }; // r30 = 0 : NOP
```

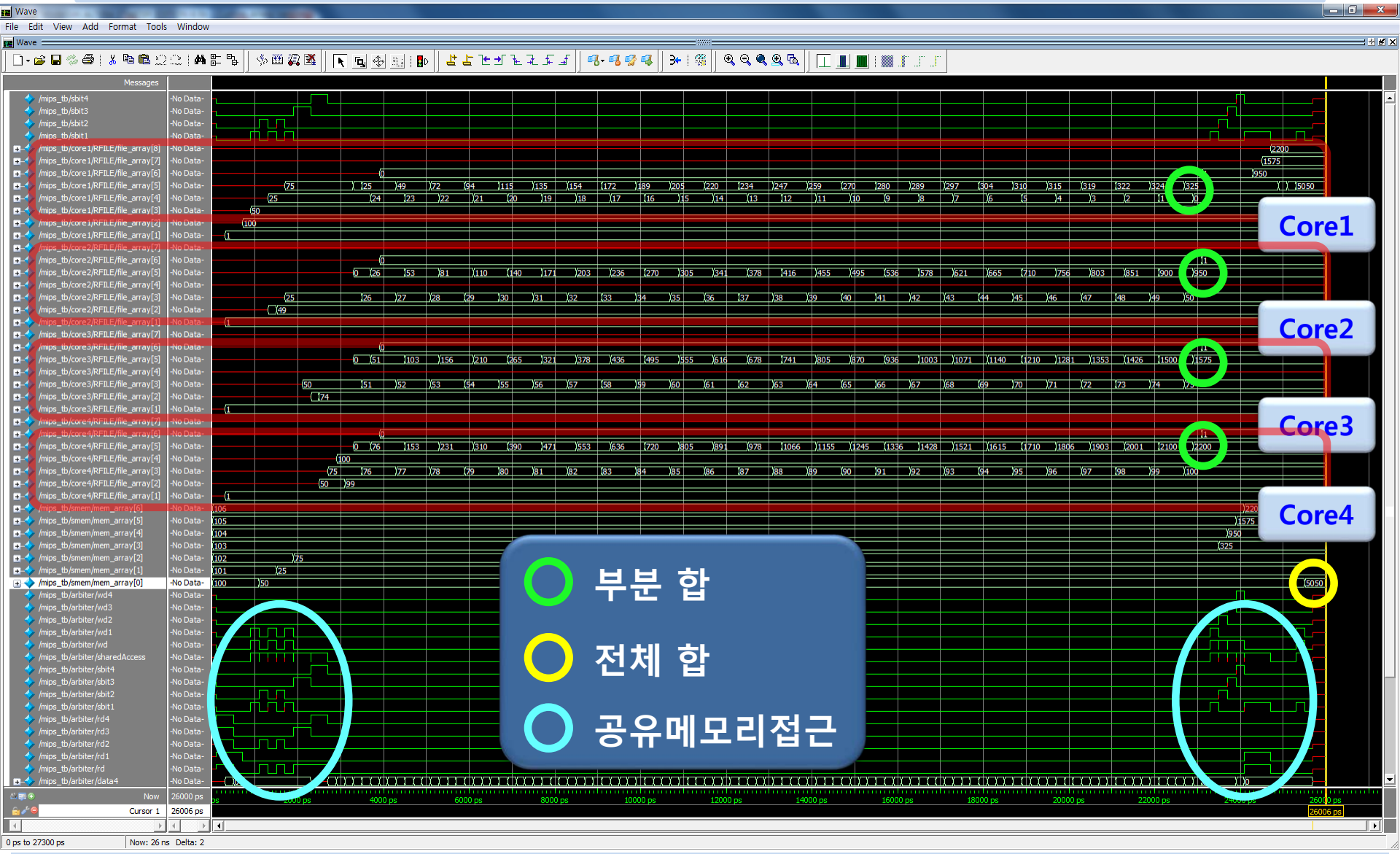
Core3

```
        5'd0 : data_out = { 6'd35, 5'd0, 5'd1, 16'd0 };              // lw $1, 0($0)   r1=1
        // 데이터 동기화를 위한 NOPs
        5'd1 : data_out = { 6'd0,  5'd0, 5'd0, 5'd30, 5'd0, 6'd32 }; // r30 = 0 : NOP
        5'd2 : data_out = { 6'd0,  5'd0, 5'd0, 5'd30, 5'd0, 6'd32 }; // r30 = 0 : NOP
        5'd3 : data_out = { 6'd0,  5'd0, 5'd0, 5'd30, 5'd0, 6'd32 }; // r30 = 0 : NOP
        5'd4 : data_out = { 6'd0,  5'd0, 5'd0, 5'd30, 5'd0, 6'd32 }; // r30 = 0 : NOP
        5'd5 : data_out = { 6'd0,  5'd0, 5'd0, 5'd30, 5'd0, 6'd32 }; // r30 = 0 : NOP
        5'd6 : data_out = { 6'd0,  5'd0, 5'd0, 5'd30, 5'd0, 6'd32 }; // r30 = 0 : NOP
        5'd7 : data_out = { 6'd0,  5'd0, 5'd0, 5'd30, 5'd0, 6'd32 }; // r30 = 0 : NOP
        5'd8 : data_out = { 6'd35, 5'd0, 5'd3, 16'd128 };            // r3 <- SharedMEM[0] = 50
        5'd9 : data_out = { 6'd35, 5'd0, 5'd2, 16'd136 };            // r2 <- SharedMEM[2] = 75
        5'd10: data_out = { 6'd0,  5'd2, 5'd1, 5'd2, 5'd0, 6'd34 };  // add $2, $2, #1  r2 = r2 - r1
        5'd11: data_out = { 6'd0,  5'd0, 5'd0, 5'd30, 5'd0, 6'd32 }; // r30 = 0 : NOP
        5'd12: data_out = { 6'd0,  5'd0, 5'd0, 5'd30, 5'd0, 6'd32 }; // r30 = 0 : NOP
        5'd13: data_out = { 6'd0,  5'd0, 5'd0, 5'd30, 5'd0, 6'd32 }; // r30 = 0 : NOP

        // 26 + 27 + ... + 50
        5'd14: data_out = { 6'd0,  5'd0, 5'd0, 5'd5, 5'd0, 6'd32 };  // add $5, $0, $0  r5 = 0
        5'd15: data_out = { 6'd0,  5'd3, 5'd1, 5'd3, 5'd0, 6'd32 };  // add $3, $3, #1  r3 = r3 + r1
        5'd16: data_out = { 6'd0,  5'd3, 5'd5, 5'd5, 5'd0, 6'd32 };  // add $5
        5'd17: data_out = { 6'd0,  5'd2, 5'd3, 5'd6, 5'd0, 6'd42 };  // slt $6
        5'd18: data_out = { 6'd4,  5'd6, 5'd0, -16'd4 };             // beq $6
        5'd19: data_out = { 6'd0,  5'd0, 5'd0, 5'd30, 5'd0, 6'd32 }; // r30 =

        //
        5'd20: data_out = { 6'd0,  5'd0, 5'd0, 5'd30, 5'd0, 6'd32 }; // r30 =
        5'd21: data_out = { 6'd43, 5'd0, 5'd5, 16'd148 };            // 결과 저
```

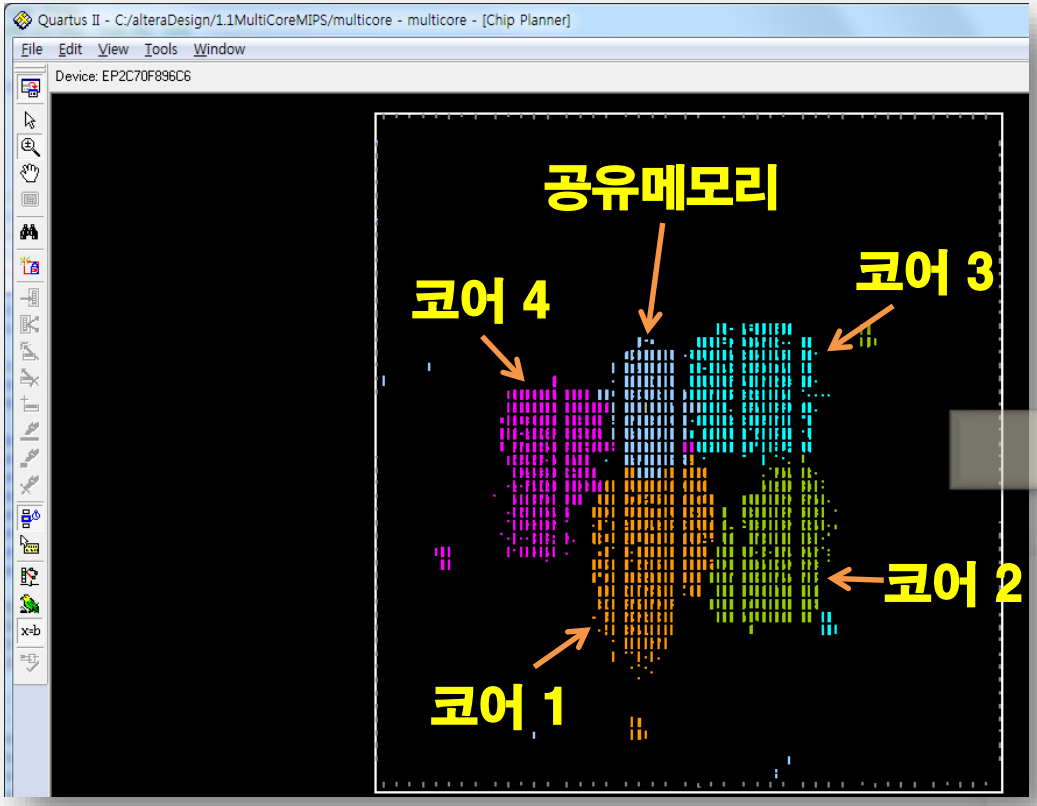
Core4

```
        5'd0 : data_out = { 6'd35, 5'd0, 5'd1, 16'd0 };              // lw $1, 0($0)   r1=1
        // 데이터 동기화를 위한 NOPs
        5'd1 : data_out = { 6'd0,  5'd0, 5'd0, 5'd30, 5'd0, 6'd32 }; // r30 = 0 : NOP
        5'd2 : data_out = { 6'd0,  5'd0, 5'd0, 5'd30, 5'd0, 6'd32 }; // r30 = 0 : NOP
        5'd3 : data_out = { 6'd0,  5'd0, 5'd0, 5'd30, 5'd0, 6'd32 }; // r30 = 0 : NOP
        5'd4 : data_out = { 6'd0,  5'd0, 5'd0, 5'd30, 5'd0, 6'd32 }; // r30 = 0 : NOP
        5'd5 : data_out = { 6'd0,  5'd0, 5'd0, 5'd30, 5'd0, 6'd32 }; // r30 = 0 : NOP
        5'd6 : data_out = { 6'd0,  5'd0, 5'd0, 5'd30, 5'd0, 6'd32 }; // r30 = 0 : NOP
        5'd7 : data_out = { 6'd0,  5'd0, 5'd0, 5'd30, 5'd0, 6'd32 }; // r30 = 0 : NOP
        5'd8 : data_out = { 6'd0,  5'd0, 5'd0, 5'd30, 5'd0, 6'd32 }; // r30 = 0 : NOP
        5'd9 : data_out = { 6'd0,  5'd0, 5'd0, 5'd30, 5'd0, 6'd32 }; // r30 = 0 : NOP
        5'd10: data_out = { 6'd35, 5'd0, 5'd2, 16'd128 };            // r2 <- SharedMEM[0] = 50
        5'd11: data_out = { 6'd35, 5'd0, 5'd3, 16'd136 };            // r3 <- SharedMEM[2] = 75
        5'd12: data_out = { 6'd0,  5'd2, 5'd2, 5'd4, 5'd0, 6'd34 };  // add $4, $2, $2  r4 = r2 + r2
        5'd13: data_out = { 6'd0,  5'd4, 5'd1, 5'd2, 5'd0, 6'd34 };  // add $2, $2, #1  r2 = r4 - r1

        // 26 + 27 + ... + 50
        5'd14: data_out = { 6'd0,  5'd0, 5'd0, 5'd5, 5'd0, 6'd32 };  // add $5, $0, $0  r5 = 0
        5'd15: data_out = { 6'd0,  5'd3, 5'd1, 5'd3, 5'd0, 6'd32 };  // add $3, $3, #1  r3 = r3 + r1
        5'd16: data_out = { 6'd0,  5'd3, 5'd5, 5'd5, 5'd0, 6'd32 };  // add $5, $5, $3  r5 = r5 + r3
        5'd17: data_out = { 6'd0,  5'd2, 5'd3, 5'd6, 5'd0, 6'd42 };  // slt $6, $2, $3  if r2 < r3 ?
        5'd18: data_out = { 6'd4,  5'd6, 5'd0, -16'd4 };             // beq $6, $0, -4  if not, go back 3
        5'd19: data_out = { 6'd0,  5'd0, 5'd0, 5'd30, 5'd0, 6'd32 }; // r30 = 0 : NOP

        5'd20: data_out = { 6'd0,  5'd0, 5'd0, 5'd30, 5'd0, 6'd32 }; // r30 = 0 : NOP
        5'd21: data_out = { 6'd0,  5'd0, 5'd0, 5'd30, 5'd0, 6'd32 }; // r30 = 0 : NOP
        5'd22: data_out = { 6'd43, 5'd0, 5'd5, 16'd152 };            // r5 <- SharedMEM[3] = 1+2+...+25
        5'd23: data_out = { 6'd0,  5'd0, 5'd0, 5'd30, 5'd0, 6'd32 }; // r30 = 0 : NOP
```

1 ~ 100 더하기@QuadCore



칩 배치 설계 (Altera Chip Planner)



다운로드