Computer Architecture

Project : 난수 생성 특화 CPU 설계

12191529 장준영

12211472 김현민

12211785 송상윤

12236553 안효리

목차

- Target & Motivation
- Base ISA 및 설계 철학
- Instruction-set extension
- Summary

Target & Motivation

- 난수: 통계/시뮬레이션/암호/네트워킹 등에서 많이 사용.
- 난수를 생성하기 위한 알고리즘
: 선형합동법, Mersenne Twister, XORshift, AES-CTR 등

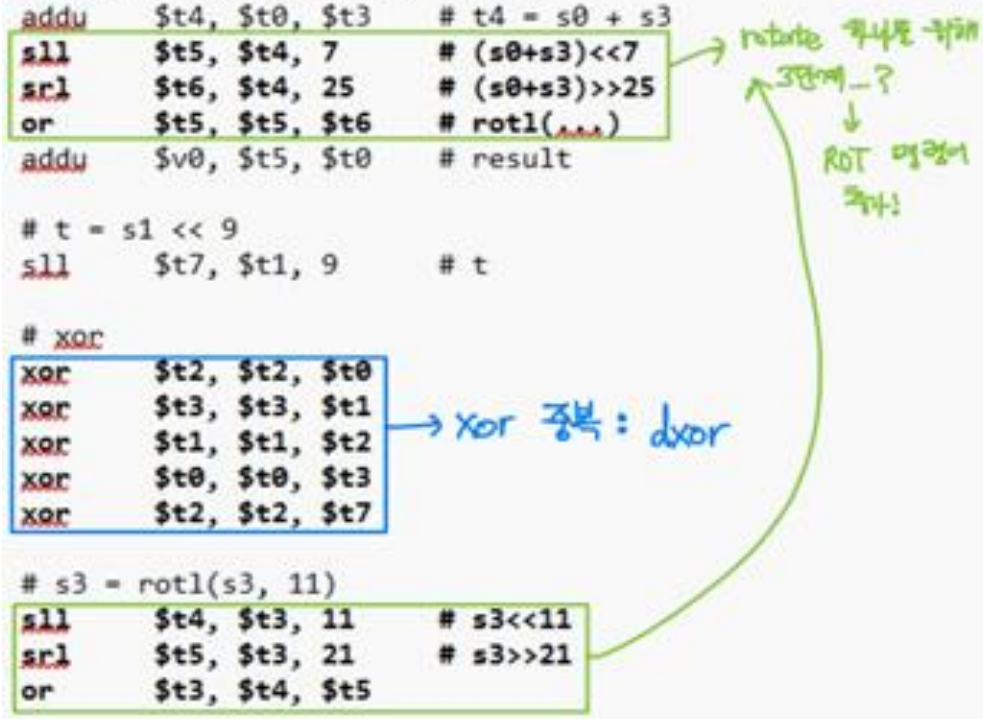
>> XOR 연산이 반복적으로 사용하고, 선형성을 없애기 위해 shift, rotate 등을 사용하는 **XORshift, xoshiro** 알고리즘을 타겟으로 잡음.

∴ 목표: 난수 생성 연산에서의 부분적 가속.

Target & Motivation

```
<xorshift32>
입력 $a0=x, 출력 $v0

move    $t0, $a0          # x

sll     $t1, $t0, 13      # x<<13
xor     $t0, $t0, $t1     # x ^= (x<<13)

srl     $t1, $t0, 17      # x>>17 (logical)
xor     $t0, $t0, $t1

sll     $t1, $t0, 5       # x<<5
xor     $t0, $t0, $t1

move    $v0, $t0
```

```
<xoshiro128+>
입력 $a0 = &s[0] (s[0]=s0, s[1]=s1, s[2]=s2, s[3]=s3)
반환 $v0 = result

lw      $t0, 0($a0)      # s0
lw      $t1, 4($a0)      # s1
lw      $t2, 8($a0)      # s2
lw      $t3, 12($a0)     # s3

# result = rotl(s0 + s3, 7) + s0
addu    $t4, $t0, $t3    # t4 = s0 + s3
sll     $t5, $t4, 7      # (s0+s3)<<7
srl     $t6, $t4, 25     # (s0+s3)>>25
or      $t5, $t5, $t6    # rotl(...)
addu    $v0, $t5, $t0    # result

# t = s1 << 9
sll     $t7, $t1, 9      # t

# xor
xor     $t2, $t2, $t0
xor     $t3, $t3, $t1
xor     $t1, $t1, $t2
xor     $t0, $t0, $t3
xor     $t2, $t2, $t7

# s3 = rotl(s3, 11)
sll     $t4, $t3, 11     # s3<<11
srl     $t5, $t3, 21     # s3>>21
or      $t3, $t4, $t5

# store state back
sw      $t0, 0($a0)      # s0'
sw      $t1, 4($a0)      # s1'
sw      $t2, 8($a0)      # s2'
sw      $t3, 12($a0)     # s3'
```



Base ISA 및 설계 철학

- MiniMIPS ISA 기반 위에 명령어 추가.

∵ 기본적인 CPU 동작 구현 + 타겟 알고리즘을 위한 특화 명령어만 추가.

- 통계 연산, 난수 생성 분야 특화 명령어 추가.

“작지만 목적에 맞게 최적화된 CPU”

Base ISA 및 설계 철학

Instruction	Usage
Load upper immediate	lui rt,imm
Add	add rd,rs,rt
Subtract	sub rd,rs,rt
Set less than	slt rd,rs,rt
Add immediate	addi rt,rs,imm
Set less than immediate	slti rd,rs,imm
AND	and rd,rs,rt
OR	or rd,rs,rt
XOR	xor rd,rs,rt
NOR	nor rd,rs,rt
AND immediate	andi rt,rs,imm
OR immediate	ori rt,rs,imm
XOR immediate	xori rt,rs,imm
Load word	lw rt,imm(rs)
Store word	sw rt,imm(rs)
Jump	j L
Jump register	jr rs
Branch less than 0	bltz rs,L
Branch equal	beq rs,rt,L
Branch not equal	bne rs,rt,L

Instruction	Usage
Move from Hi	mfhi rd
Move from Lo	mflo rd
Add unsigned	addu rd,rs,rt
Subtract unsigned	subu rd,rs,rt
Multiply	mult rs,rt
Multiply unsigned	multu rs,rt
Divide	div rs,rt
Divide unsigned	divu rs,rt
Add immediate unsigned	addiu rs,rt,imm
Shift left logical	sll rd,rt,sh
Shift right logical	srl rd,rt,sh
Shift right arithmetic	sra rd,rt,sh
Shift left logical variable	sllv rd,rt,rs
Shift right logical variable	srlv rd,rt,rs
Shift right arith variable	srav rd,rt,rs
Load byte	lb rt,imm(rs)
Load byte unsigned	lbu rt,imm(rs)
Store byte	sb rt,imm(rs)
Jump and link	jal L
System call	syscall

MiniMIPS ISA (출처: 컴퓨터구조론 강의노트 COA-04 Information_System)

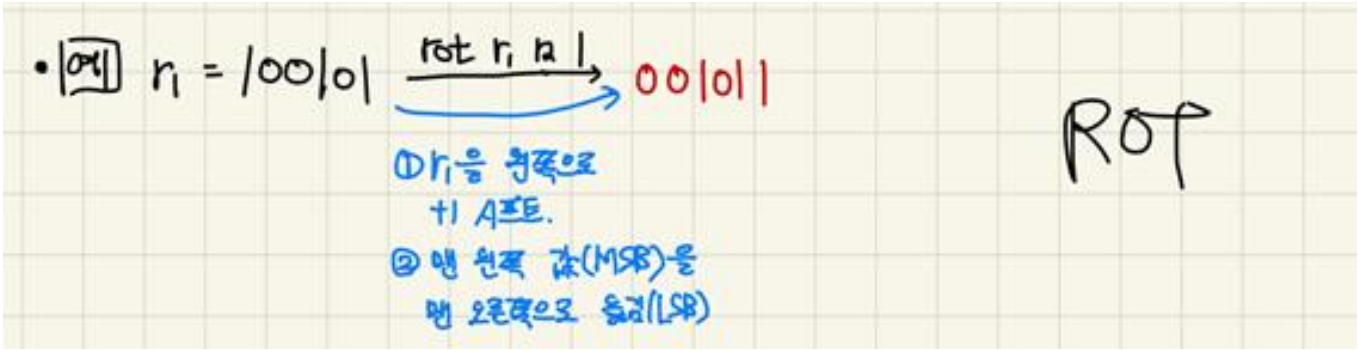
ISA extension

- rot : 순환 시프트(rotate)
- slxor : Shift-Left + XOR
- srxor : Shift-Right + XOR
- dxor : 두 레지스터 쌍을 동시에 XOR

ISA extension (1) : rot

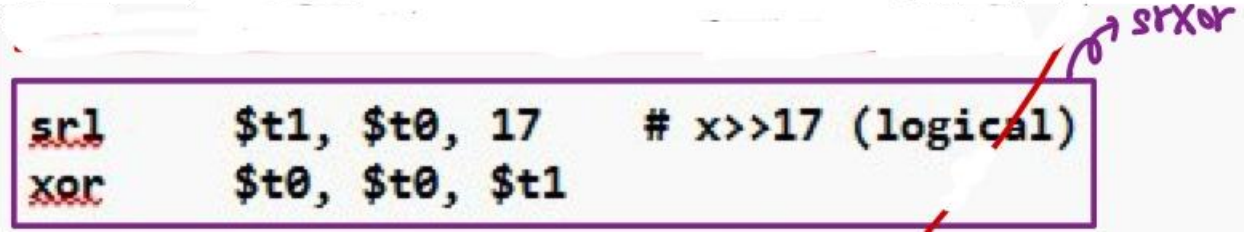
rot : 순환 시프트(rotate)

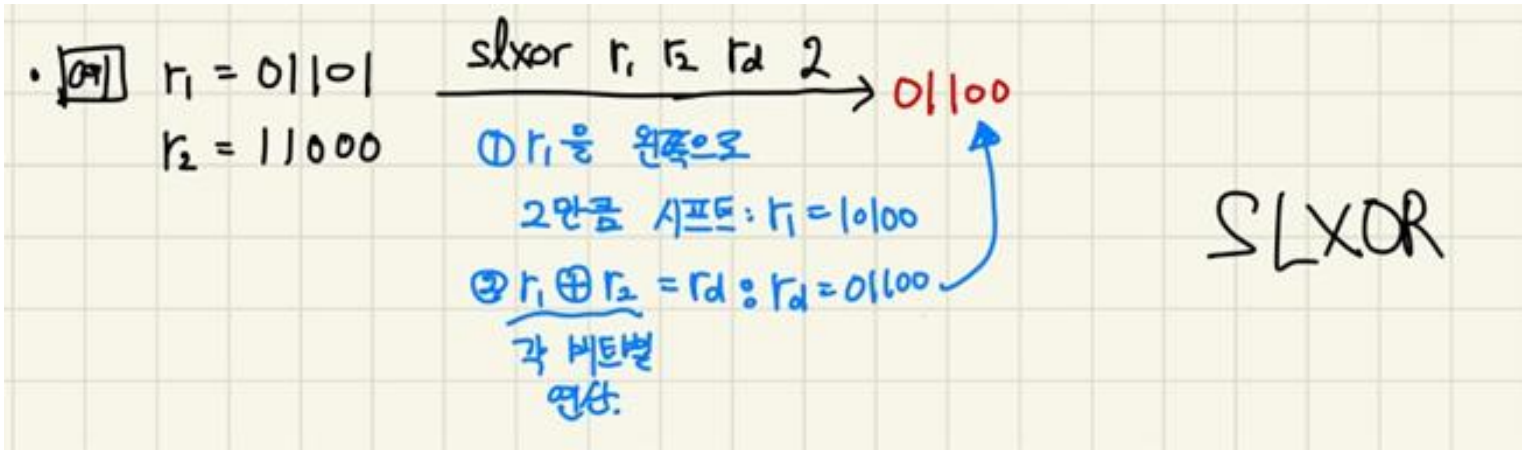
- 비트 회전 연산은 비트들을 한쪽으로 이동시키면서 맨 끝으로 밀려난 비트는 반대편 맨 앞으로 채워 넣는 연산
- 기존 MiniMIPS에서는 명령어 3개의 조합으로 rotate 연산을 구현
- 난수 생성에서 많이 쓰이는 rotate연산 최적화를 위해 명령어 생성



ISA extension (2) : slxor

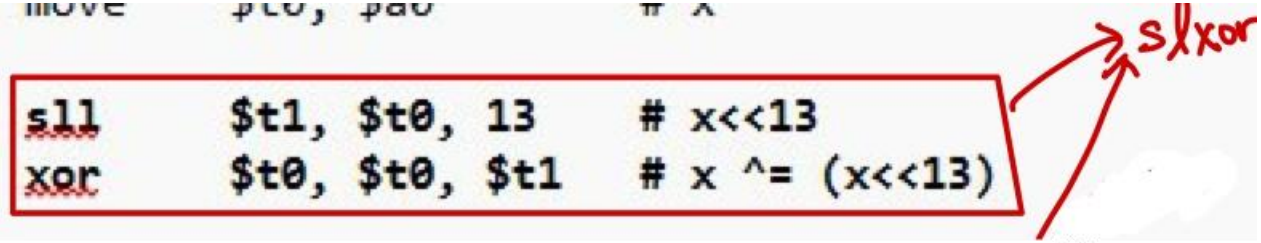
- slxor : Shift-Left + XOR
- 도입 목적
  : 2개의 연산이 계속해서 반복되므로 이를 하나의 명령어로 만들어 속도를 개선하고 단순하게 만들기 위함.

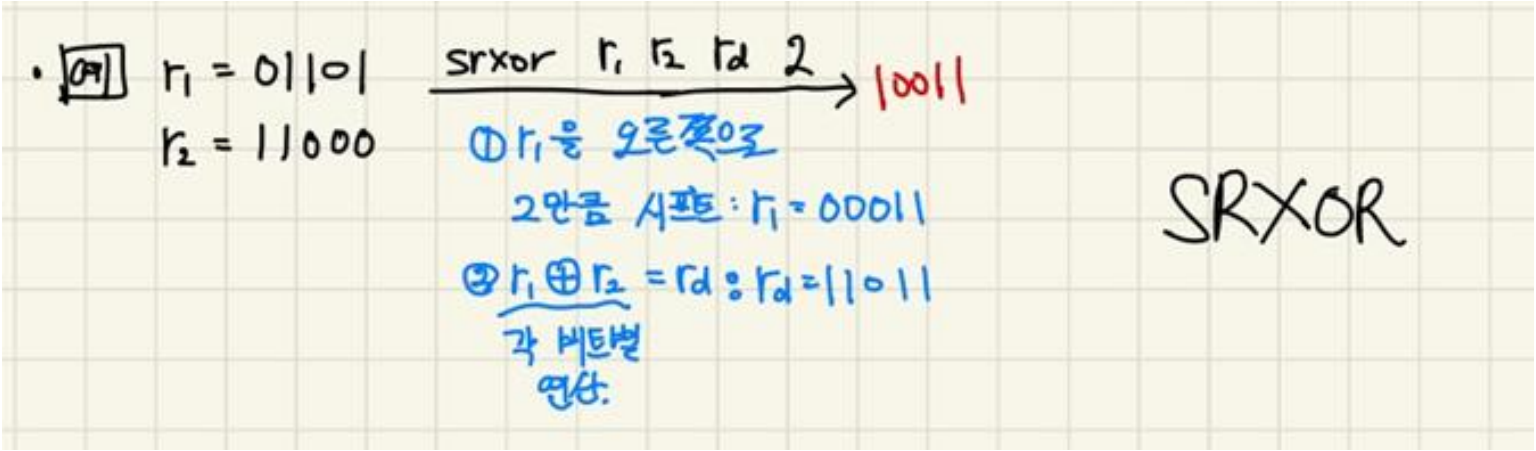




ISA extension (3) : srxor

- srxor : Shift-Right + XOR
- 도입 목적 : slxor과 같음. 방향만 반대



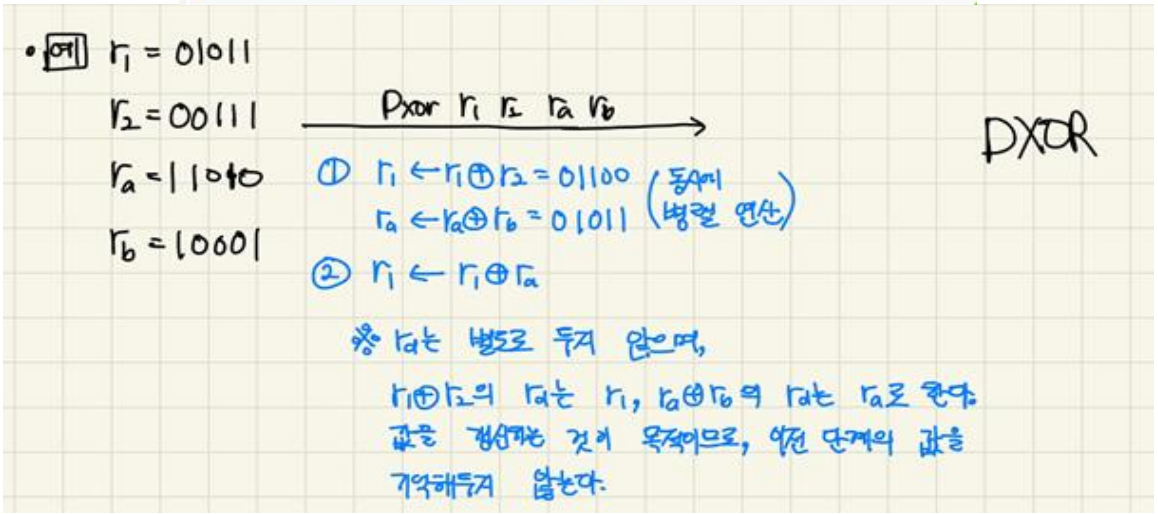


ISA extension (4) : dxor

- dxor: 두 레지스터 쌍을 동시에 XOR.
- 도입 목적:
  여러 번의 XOR 연산을 한 번에 처리하기 위함.

```
# xor
xor     $t2, $t2, $t0
xor     $t3, $t3, $t1
xor     $t1, $t1, $t2
xor     $t0, $t0, $t3
xor     $t2, $t2, $t7
```

→ xor 중복 : dxor



ISA extension 정리

Name	Fields	비고
ROT	op | r1 | rs | rd | | ext	방향은 왼쪽, rs 값 만큼 회전 (R type)
SLXOR	op | r1 | r2 | rd | im_var | ext	Carry 폐기, 0 채우기 (R type)
SRXOR	op | r1 | r2 | rd | im_var | ext	Carry 폐기, 0 채우기 (R type)
DXOR	op | r1 | r2 | ra | rb | ext	r1⊕r2 → r1 | ra⊕rb → ra (새로운 type)

Summary & 이후 목표

- 난수.비트조작에 특화된 R-타입 명령어 4개 구성
- MiniMIPS ISA 위에 특화된 명령어 추가해서 CPU 구현.
- 자세한 설계 방법 및 기존 계획은, 이후 배울 내용에 따라 일부 수정하고자 함.