

How a processor can permute n bits in $O(1)$ cycles



Ruby Lee,
Zhijie Shi, Xiao Yang

Princeton Architecture Lab for Multimedia and Security (PALMS)
Department of Electrical Engineering
Princeton University

IEEE Hot Chips 14, August 2002

Motivation



- Secure information processing increases in importance in interconnected world
- Word-oriented microprocessors today can handle cryptography algorithms well, except for:
 - Bit-level permutations
 - Multi-word arithmetic
- The larger architectural question:
 - Can a word-oriented processor handle complex bit-level operations within the word efficiently?

Today - microprocessor or ASIC

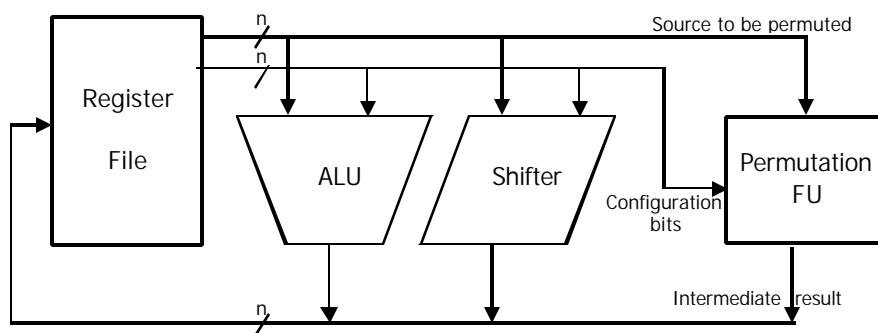


- Logic Operations
 - MASK-Gen/AND/SHIFT/OR → $4n$ instructions
 - EXTRACT/DEPOSIT → $2n$ instructions
- Table lookup
 - small set of fixed permutations only
 - 8x2KB tables, about 32 instructions for 64 bits permutation
- Subword permutation instructions for multimedia
 - Works on 8-bit or larger subwords
- ASIC
 - permutation very fast in hardware, BUT
 - small set of fixed permutations only

Goal: add new Permutation Functional Unit to Processor



Achieve any one of $n!$ permutations in $\log(n)$ instructions



Initial Problem Definition



- Efficient bit permutation instructions for arbitrary permutations of n bits
 - Focus on $n = 32$ or 64 (word sizes)
 - Standard instruction format and datapaths
 - 2 reads, 1 write per instruction
 - No extra state (to save and restore)
 - Single cycle, simple hardware
 - in $\log(n)$ instructions - optimal
 - Number of different n -bit permutation = $n!$
 $\log(n!) \approx n \log(n) \quad (n > 0)$
 - $n \log(n)$ bits needed to specify an arbitrary permutation

Outline



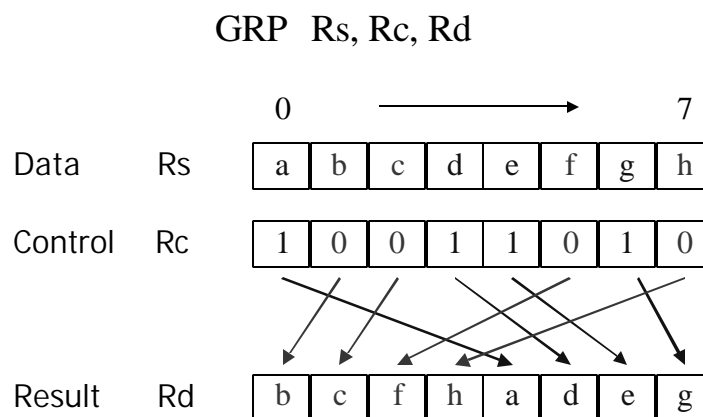
- Permute n bits: from $O(n)$ to $O(\log(n))$ instructions
 - ISA definitions
 - Chip/Circuit Implementations
 - Performance, Cycletime, Versatility
- Permute n bits: from $O(\log(n))$ to $O(1)$ cycles
- Conclusion

Alternative permutation methods

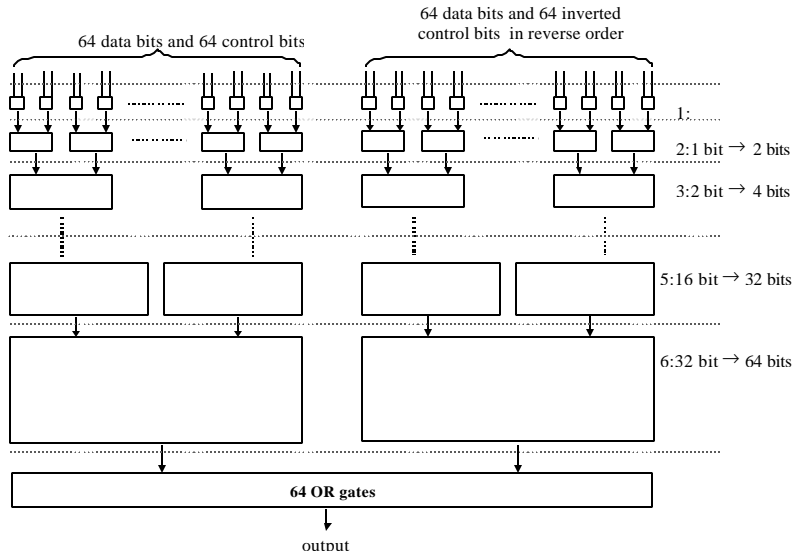


- to reduce $O(n)$ to $O(\log n)$ instructions for achieving any one of $n!$ permutations
- Partitioning
 - GRP
- Building “virtual” interconnection networks
 - CROSS ($\log(n)$ types of stages)
 - OMFLIP (2 types of stages)
- Select source bit by its numeric index
 - PPERM
 - SWPERM and SIEVE

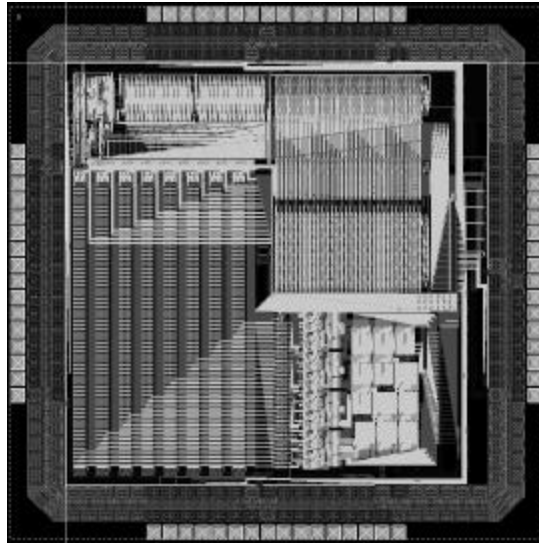
8-bit GRP operation



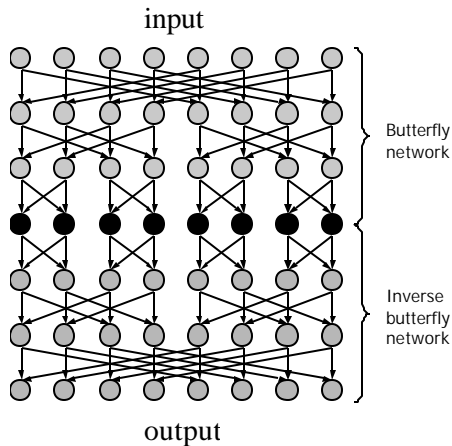
GRP64 Implementation



Chip with Permutation Unit (GRP)

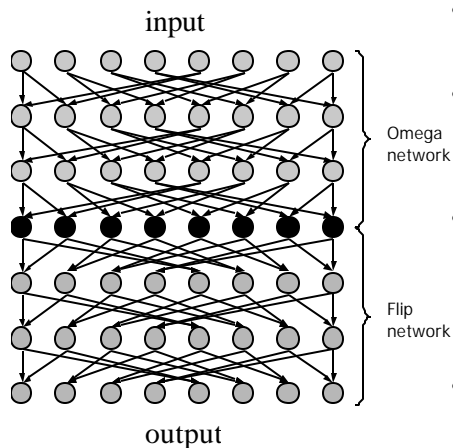


8-bit CROSS instruction ¾ building a virtual Benes Network



- perform any 2 butterfly stages in one instruction
- Performs any n-bit permutation with $2\log(n)$ stages
- $\log(n)$ different types of stages
- Scalable for subword permutation
- Shortest latency

8-bit OMFLIP ¾ building a virtual Omega-Flip Network



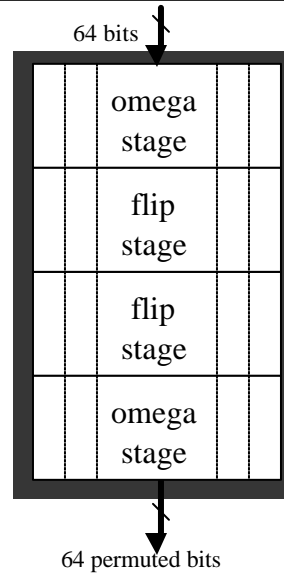
- perform 2 omega or flip stages in one instruction
- Performs any n-bit permutation with $2\log(n)$ stages
- Only 2 different types of stages
- Scalable for subword permutation
- Smallest area for a permutation unit

An OMFLIP Implementation

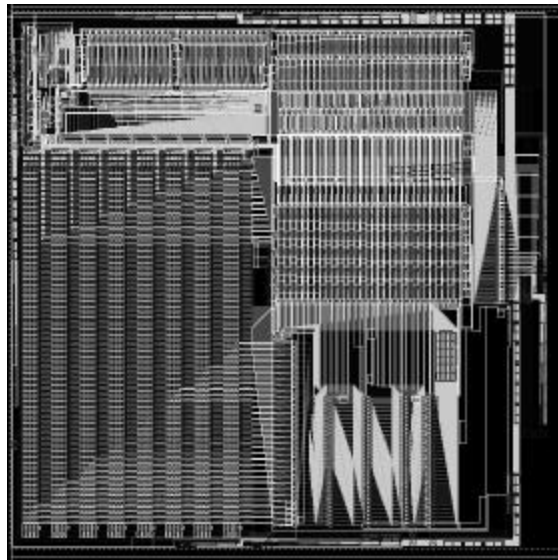


- To implement any 2 combinations of Omega or Flip stages, it is enough to implement a circuit with only 4 stages, 2 omega stages, 2 flip stages
- This allows 00, FF, OF and FO combinations
- *Other circuit organizations also possible, e.g., O-F-O-F, F-O-F-O and F-O-O-F*

by passing connections



Chip with Permutation Unit (OMFLIP)



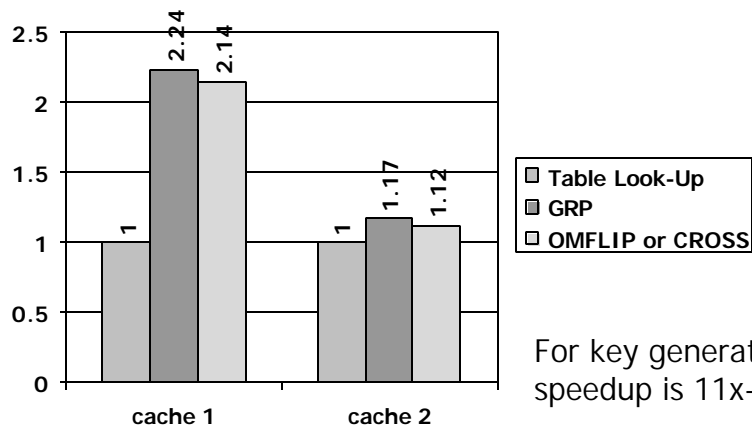
Comparison



Maximum Number of Instructions Required for Any Permutation

	Current ISA	Table lookup	GRP	OMFLIP or CROSS
Bit permutation, n elements, each 1-bit	$\Theta(n)$	$\Theta(n)$	$\log(n)$	$\log(n)$
Subword permutation, n/k elements, each k-bit	$\Theta(n/k)$	$\Theta(n)$	$\log(n/k)$	$\log(n/k)$

Speedup of DES



For key generation, speedup is 11x-16X !

Cache 1: one-level cache, 16KB (50 cycles miss penalty).

Cache 2: two-level cache, L1: 16KB (10 cycles miss penalty), L2: 256KB (50 cycles)

Speedup for sorting 64 elements using GRP instruction



Subword size	4 bits	8 bits	16 bits
vs. Bubble sort	408.3	128.9	43.7
vs. Selection sort	272.7	86.1	29.2
vs. Quick sort	94.4	29.8	10.1

Demonstrates versatility of GRP instructions for sorting as well as permutations.

How to execute $\log(n)$ instructions in $O(1)$ cycles?



Instruction sequence to permute 64 bits:

```
OMFLIP,00 R1,R2,R10
OMFLIP,00 R10,R3,R10
OMFLIP,00 R10,R4,R10
OMFLIP,ff R10,R5,R10
OMFLIP,ff R10,R6,R10
OMFLIP,ff R10,R7,R10
...
```

- RISC ISA constraint of instructions with only 2 operands
- n-bit permutation needs $1+\log(n)$ operands
- Supplying these operands results in register data dependencies
- But 7 operands could be supplied in 4 RISC instructions rather than 6?

Leverage microarchitecture features in 2-way superscalar processors



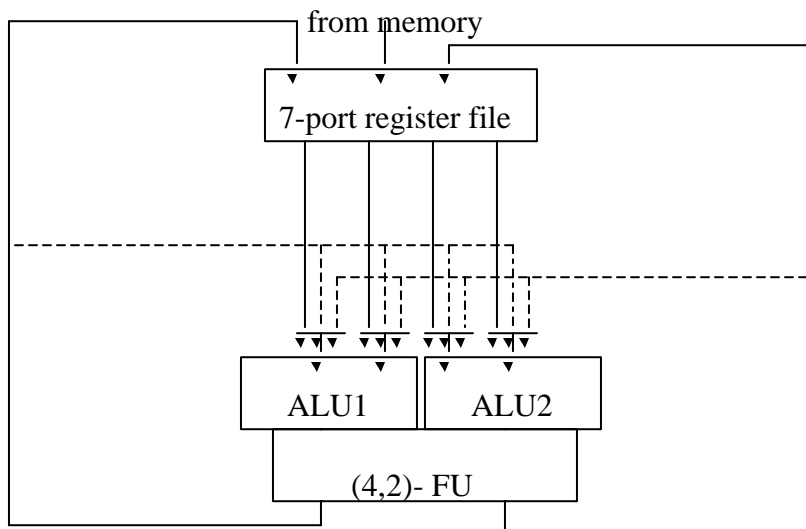
Original instruction sequence to permute 64 bits:

- Enable “Data-rich” functional units utilizing existing parallel register ports and data buses
- Replace 6 instructions with 4 (ISA or microarchitecture)

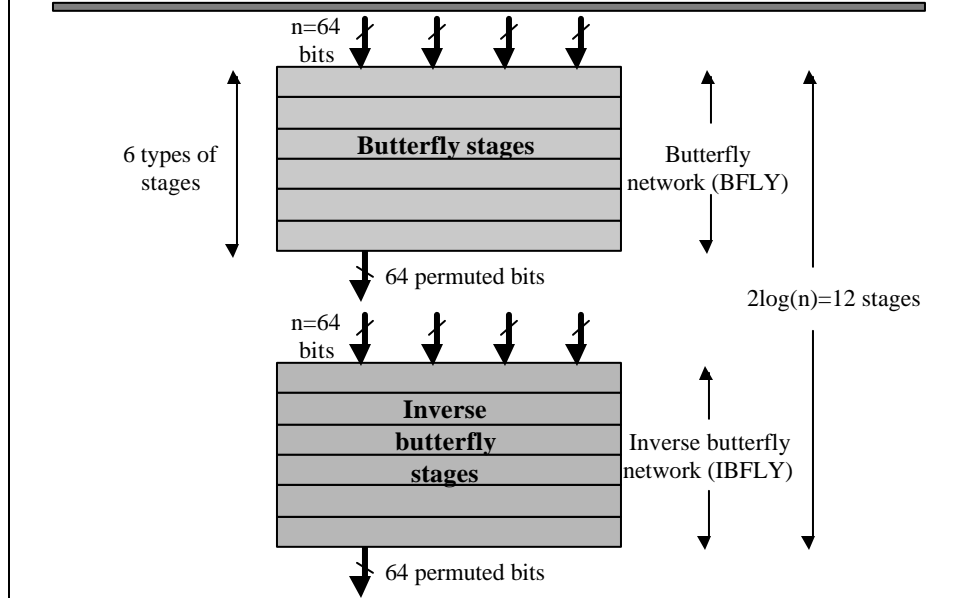
```
OMFLIP,oo R1,R2,R10
OMFLIP,oo R10,R3,R10
OMFLIP,oo R10,R4,R10
OMFLIP,ff R10,R5,R10
OMFLIP,ff R10,R6,R10
OMFLIP,ff R10,R7,R10
```

```
OMFLIP,oo R1,R2,R10
OMcont   R4,R3,R10
OMFLIP,ff R10,R5,R10
OMcont   R7,R6,R10
```

2-way Superscalar with a (4,2) Data-rich Functional Unit



Two (4,1) functional units, each $\log(n)$ stages (Butterfly is faster than Omega-flip)



Performing any permutation of n bits with 2 cycles latency, 1 cycle thruput



- Consider $n=64$ bits
- Implement 2 permutation functional units, each with $\log(n)$ stages
 - e.g., 6-stage Butterfly network,
6-stage InverseButterfly network
- Use Data-rich (4,1) functional unit leveraging datapaths of 2-way superscalar microarchitecture
 - Replace former $\log(n)=6$ instructions by 4 instructions via ISA or microarchitecture
- Execute these 4 instructions, two at a time
 - 2 cycles latency but 1 cycle thruput
- Can achieve any one of $n!$ permutations at the rate of one per cycle
 - different permutation possible every cycle

Conclusions



- Very fast, easily implementable, general-purpose permutation instructions for any processor
 - Radical speedup: from $O(n)$ to $O(\log n)$ instructions
 - Latest result: down to $O(1)$ cycles !!
 - Can achieve any one of $n!$ permutations at the rate of one per cycle
- Important applications: accelerates both secure and multimedia information processing
 - single-bit and multi-bit subword permutations
 - big speedup in current algorithms, e.g., DES
 - opens field for faster, “more secure” new algorithms
 - versatile, multi-purpose primitives, e.g., for sorting
- Validates basic word-orientation of processors even for complex bit operations within a word