MITSUBISHI ELECTRIC

Changes for the Better

Recent Topics on Symmetric Ciphers

- Security and implementation of S-box -

October 5 2006

Mitsuru Matsui

Mitsubishi Electric Corporation

Overview

- Trends of Block/Hash Primitives and Intel Processors
- Security Issues on S-box
  - *Differential cryptanalysis: Security and related open problems*
  - *Linear cryptanalysis: Security and related open problems*
- Implementation Issues on S-box
  - *Processor Architecture of Pentium and Athlon*
  - *Ordinary Implementation of AES*
  - *Bitslice Implementation of AES and Camellia*

Block/Hash primitives & Intel Processors

MITSUBISHI ELECTRIC — Changes for the Better

RED: lookup tables & logical
BLUE: arithmetic & logical

1976: DES (for hardware)

1987: RC2 (16bit), FEAL (8bit)

1989: MD2 (16bit)
1990: MD4 (32bit), Multi2 (32bit)
1991: IDEA (16bit)
1992: MD5 (32bit)
1994: RC5 (32bit)
1995: SHA-1 (32bit)
1996: MISTY1
1998: AES, RC6, Serpent, Mars, Blowfish
2000: Kasumi, Camellia, Whirlpool (64bit)
2002: SHA-2 (32,64bit)
2004: ARIA

70
75
80
85
90
95
00
05

1971: 4004 (4bit,4KB,740KHz) First processor

1974: 8080 (8bit,64KB,2MHz)

1978: 8086 (16bit,1MB,5-10MHz) Segment

1982: 80286 (16bit,16MB,6-12.5MHz) Protect mode

1985: 80386 (32bit,4GB,16-33MHz) Virtual memory

1989: 80486 (25-100MHz) on chip L1 cache

1993: Pentium (60-200MHz) Superscalar

1995: Pentium Pro (150-200MHz)

1997: Pentium II (233-1300MHz) 64-bit MMX

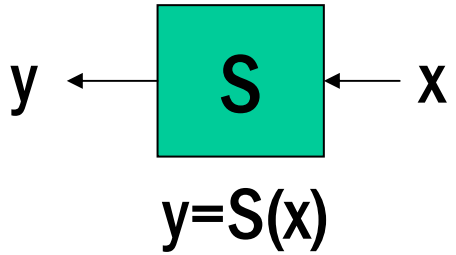
1999: Pentium III (450-1400MHz) SSE
2000: Pentium 4 (-3.4GHz) SSE2 “Northwood”

2003: Pentium M (-2.1GHz)
2004: Pentium 4 (-3.8GHz) SSE3 “Prescott” **EM64T**

2006: Core (-2.33GHz)
2006: Core2(-2.93GHz) SSE4 **EM64T**

S-Box - a lookup table -

- 6-in/4-out
  - DES design criteria unknown



- 7-in/7-out, 9-in/9-out
  - MISTY a power function over a Galois field
- 8-in/8-out
  - AES, Camellia, ARIA (block ciphers)
  - SNOW, MUGI (stream ciphers)

an inversion over Galois field $GF(2^8)$

Why an inversion over $GF(2^8)$?

(+)

- Suitable for software implementation
- Believed (but not proved) to be strongest against differential and linear cryptanalysis

(-)

- Might be weak against algebraic attacks

Differential attacks and S-box

Differential Uniformity:

$$DP_S(dx,dy) \stackrel{\text{def}}{=} \#\{x \mid S(x+dx)+S(x)=dy\}$$

Strength against differential attacks:

$$DP_S \stackrel{\text{def}}{=} max_{dx \neq 0,\, dy}\, DP_S(dx,dy)$$

(1) $DP_S \geqq 2$ *for any S.*

(2) If $S(x)=x^3$ *then* $DP_S = 2$ *for odd n.*

(3) If $S(x)=1/x$ $(S(0)=0)$ *then* $DP_S = 4$ *for even n.*

Open Problems 1

(I) Find a bijective function S over $GF(2^{2m})$ such that $DP_S=2$.

(II) Find a bijective function S over $GF(2^{2m})$ such that $DP_S=4$ and S is not linearly equivalent to an inversion.

Remark:

Probably (I) does not exist. Confirmed for $m=2$.

Linear attacks and S-box

Nonlinearity:

$$LP_S(mx,my) \overset{\text{def}}{=} |\#\{x \mid mx \bullet x = my \bullet S(x)\} - 2^{n-1}|$$

Strength against linear attacks:

$$LP_S \overset{\text{def}}{=} \max_{mx,\, my \neq 0} LP_S(mx,my)$$

(1) $LP_S \geqq 2^{(n-1)/2}$ *for any S.*

(2) If $S(x)=x^3$, *then* $LP_S = 2^{(n-1)/2}$ *for odd n.*

(3) If $S(x)=1/x$ $(S(0)=0)$, *then* $LP_S \geqq 2^{n/2}$ *for even n.*

Open Problems 2

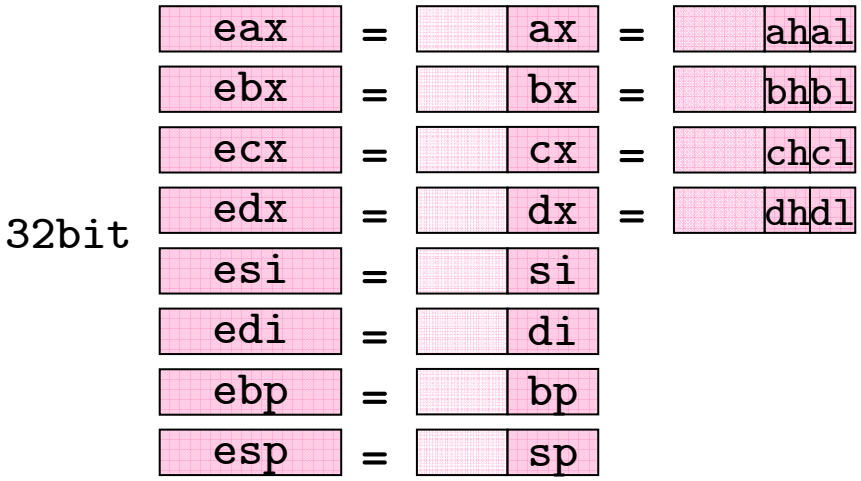
(I) *Find a function S over* $GF(2^{2m})$ *such that* $2^{(2m-1)/2} < LP_S < 2^m$.

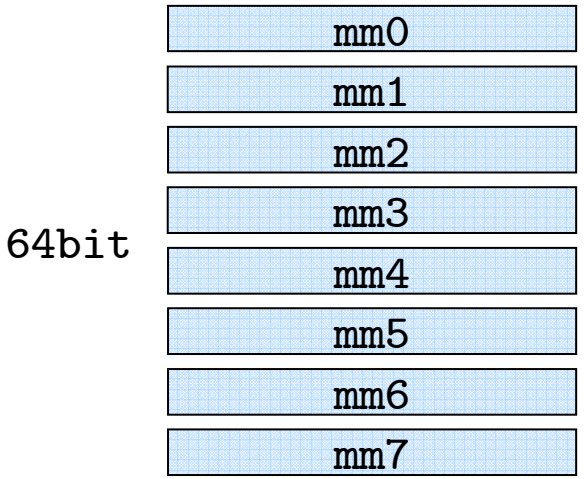
(II) *Find a bijective function S over* $GF(2^8)$ *not linearly equivalent to an inversion such that* $LP_S = 2^4$.

Remark:

Probably (I) does not exist. Confirmed for m=2.

x86 Architecture

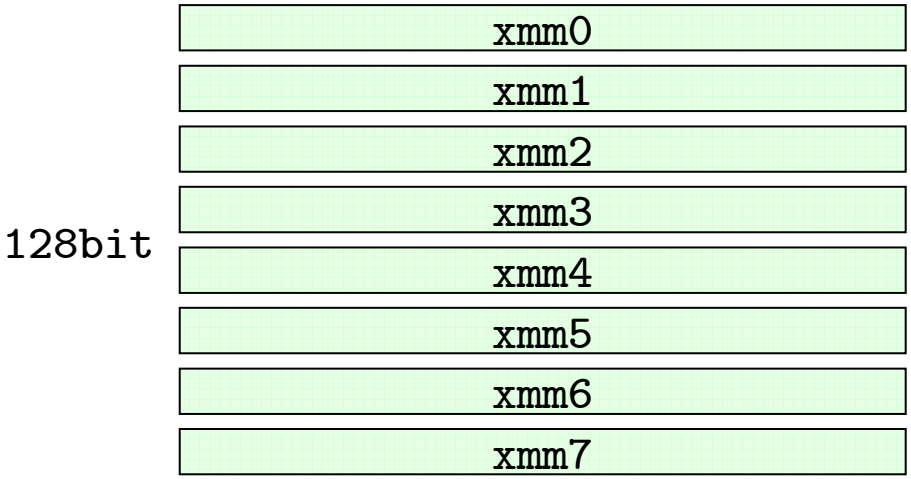




CISC Instruction Set

```
xor   eax, [esi+ebx]
add   12[ebp], al
```

destination source

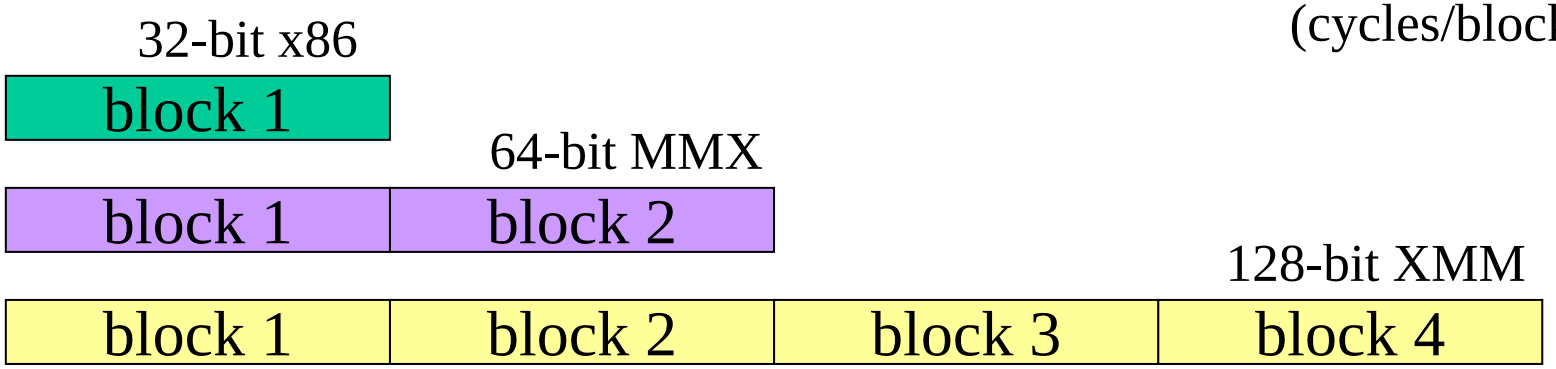


Pentium III & 4: at a glance

Encryption speed of Gladman's Serpent assembly codes optimized for P3

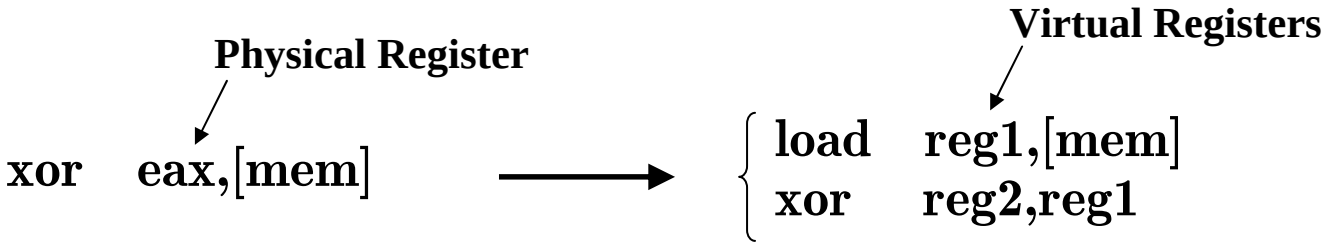
	Pentium III Coppermine	Pentium 4 Northwood	Pentium 4 Prescott
32-bit x86: Straightforward	773	1267	689
64-bit MMX: 2-block parallel	570	1052	1119
128-bit XMM: 4-block parallel	—	656	681

(cycles/block)



Micro-operations (μops)

- Pentium instructions are decomposed into RISC-style simple operations (μops) at the decoding stage
  - Intel has not published exact details on μops
- Programmers cannot direct read/write a code of micro-operations



A Pentium instruction

Corresponding μops

How to measure performance

```
xor eax,eax
cpuid
rdtsc
mov CLK1,eax
xor eax,eax
cpuid

Encryption(...,block)

xor eax,eax
cpuid
rdtsc
mov CLK2,eax
xor eax,eax
cpuid
```

```
xor eax,eax
cpuid
rdtsc
mov CLK3,eax
xor eax,eax
cpuid

/* nothing */

xor eax,eax
cpuid
rdtsc
mov CLK4,eax
xor eax,eax
cpuid
```

"Overhead"

((CLK2-CLK1) – (CLK4-CLK3)) / block

Difficulties in Measurement

- Common Implicit Assumptions
  - Should run in a constant time without interruptions
  - Should take more cycles if an interruption takes place
- These assumptions do not hold on Pentium 4 (?)

HT: Hyperthread	Northwood w/o HT	Northwood with HT
Most frequent cycles	632 cycles	636 cycles
Minimum cycles	632 cycles	600 cycles (very rare)

"Overhead" measurement results

Also Prescott Stepping 3 Revision 0 looks unstable

Advanced Encryption Standard

```
Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
   byte  state[4,Nb]

   state = in

   AddRoundKey(state, w[0, Nb-1])                    // See Sec. 5.1.4

   for round = 1 step 1 to Nr-1
      SubBytes(state)                                // See Sec. 5.1.1
      ShiftRows(state)                               // See Sec. 5.1.2
      MixColumns(state)                              // See Sec. 5.1.3
      AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
   end for

   SubBytes(state)
   ShiftRows(state)
   AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

   out = state
end
```

Figure 5. Pseudo Code for the Cipher.[1]

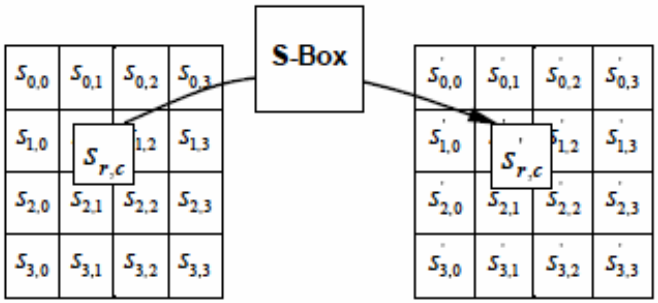


Figure 6. `SubBytes()` applies the S-box to each byte of the State.

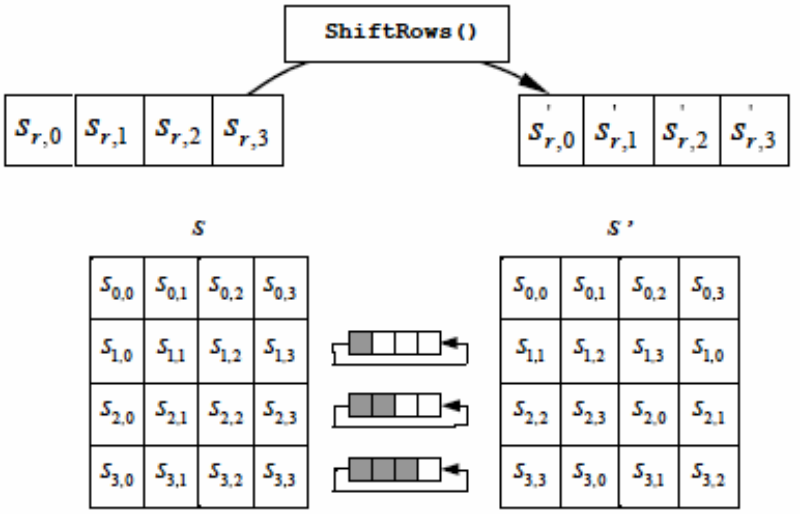


Figure 8. `ShiftRows()` cyclically shifts the last three rows in the State.

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}$$

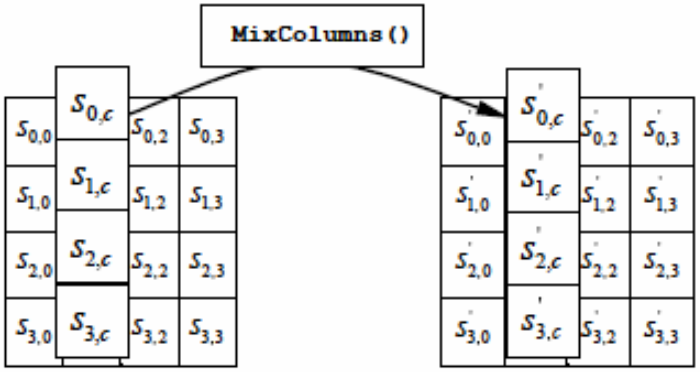


Figure 9. `MixColumns()` operates on the State column-by-colur.....

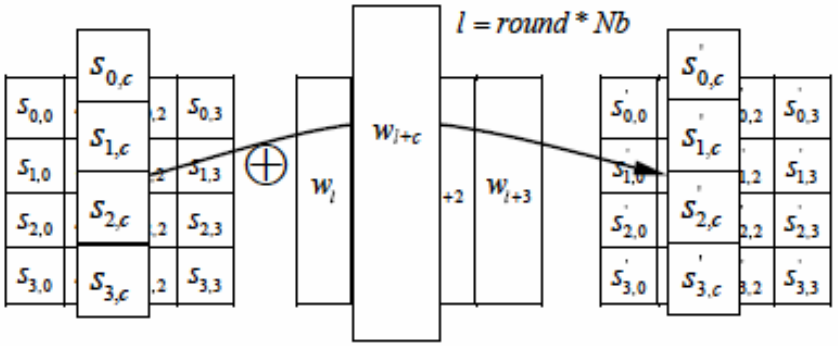
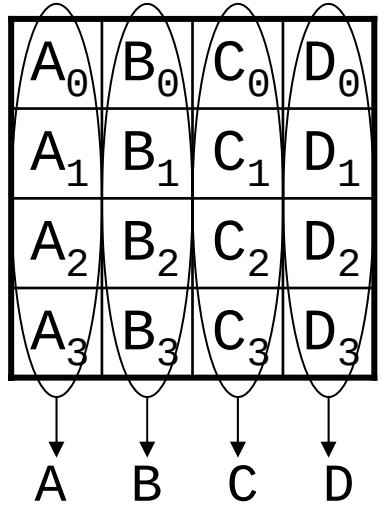


Figure 10. `AddRoundKey()` XORs each column of the State with a word from the key schedule.

One round of AES is simple

ShiftRow+SubBytes+MixColumn

$$A' = T0[A_0] \oplus T1[B_1] \oplus T2[C_2] \oplus T3[D_3]$$
$$B' = T0[B_0] \oplus T1[C_1] \oplus T2[D_2] \oplus T3[A_3]$$
$$C' = T0[C_0] \oplus T1[D_1] \oplus T2[A_2] \oplus T3[B_3]$$
$$D' = T0[D_0] \oplus T1[A_1] \oplus T2[B_2] \oplus T3[C_3]$$



AddRoundKey

$$A = A' \oplus KeyA$$
$$B = B' \oplus KeyB$$
$$C = C' \oplus KeyC$$
$$D = D' \oplus KeyD$$

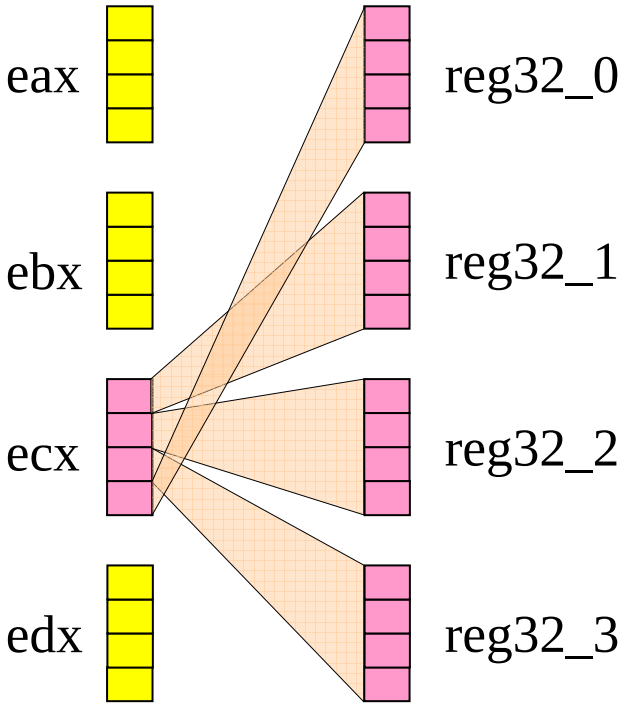
A,B,C,D,A',B',C',D':4-byte data

A_i: i-th byte of A

Ti: 1KB table (1byte->4bytes)

Another tables in the final round

AES round function in x86



ShiftRow+SubBytes+MixColumn can be done by a four-time repetition of the following sequence:

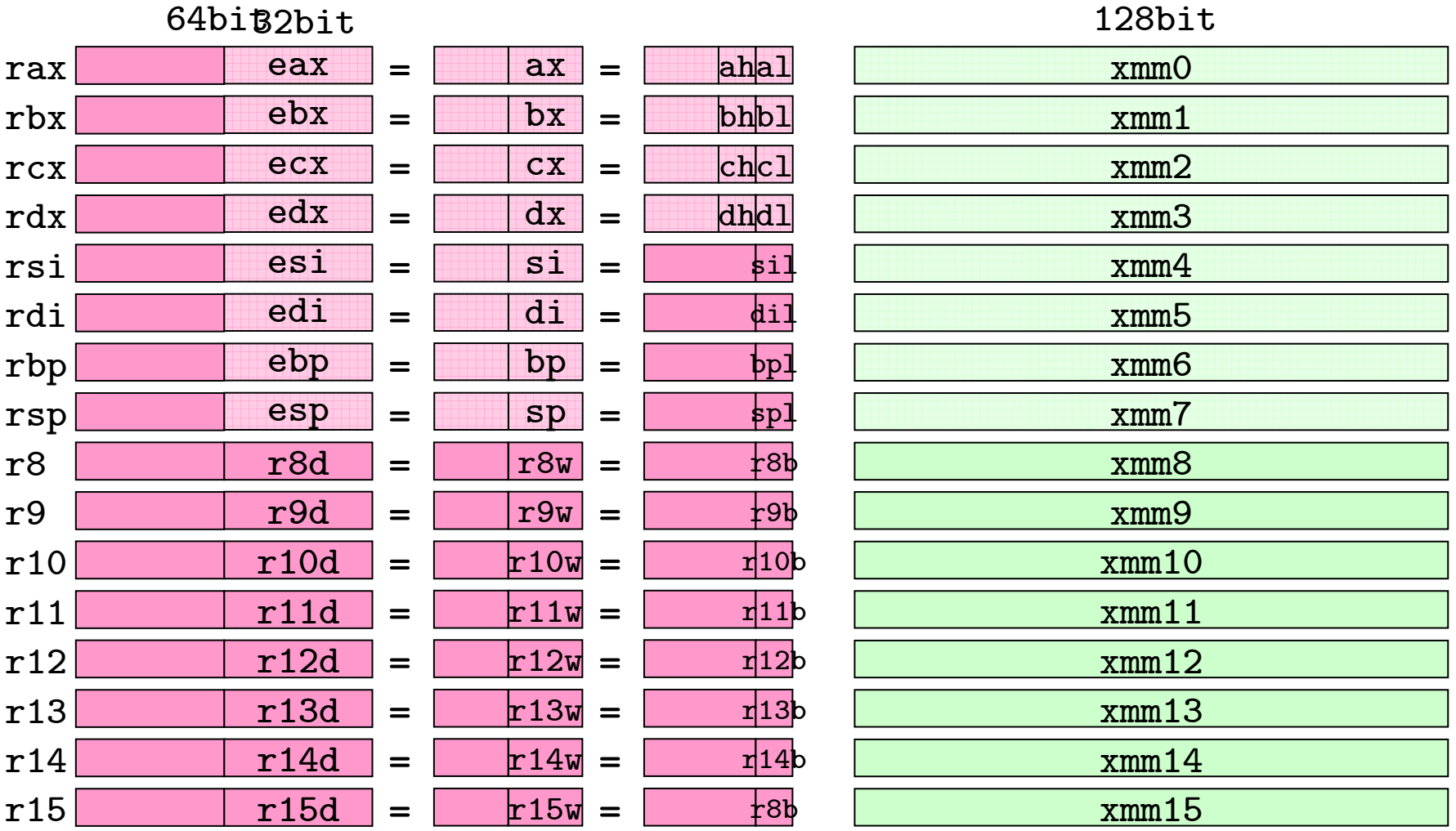
```
movzx     esi,cl
mov/xor   reg32_2,T2[esi*4]
movzx     esi,ch
mov/xor   reg32_1,T1[esi*4]
shr       ecx,16
movzx     esi,cl
mov/xor   reg32_0,T0[esi*4]
movzx     esi,ch
mov/xor   reg32_3,T3[esi*4]
```

Our implementation of AES

	Pentium III	Pentium 4 Northwood	Pentium 4 Prescott
µops / block	596	654	654
cycles / block	232	**251**	**284**
cycles / byte	14.5	**15.7**	**17.8**
µops / cycles	2.57	2.61	2.30

Slow in Prescott probably due to its high load latency

x86 vs. x64: Registers



x64: Better and Worse

(+) more registers, longer registers

(+) most instructions have a 64-bit form

ex) `rol reg32,8 => rol reg64,8`

(-) longer instruction, inefficient decoding

a prefix byte needed for an extended instruction form.

(-) a 64-bit instruction is not always fast

ex) “`shift`” and “`rotate`” on Pentium 4

Pentium 4 vs. Athlon 64

Pentium 4 (Prescott core) *up to 3.8GHz*

(+) long pipeline stages, high clock frequency

(+) instructions are cached after being decoded

(−) poorly documented, never works as Intel claims

Athlon 64 *up to 2.8GHz*

(+) high superscalability (5 uops/cycle)

(+) well documented, less frustrating for programmers

(−) its decoding stage can be a bottleneck

Instruction Latency/Throughput

Processor		Pentium 4 Prescott (EM64T)		Athlon 64 (AMD64)	
Operand Size		32-bit	64-bit	32-bit	64-bit
`mov`	`reg,[mem]`	4, 1	4, 1	3, 2	3, 2
`mov`	`reg,reg`	1, 2.88	1, 2.88	1, 3	1, 3
`add/sub`	`reg,reg`	1, 2.88	1, 2.88	1, 3	1, 3
`xor/and/or`	`reg,reg`	1, 2	1, 2	1, 3	1, 3
`shr`	`reg,imm`	1, 1.75	7, 1	1, 3	1, 3
`shl`	`reg,imm`	1, 1.75	1, 1.75	1, 3	1, 3
`ror/rol`	`reg,imm`	1, 1	7, 0.14-1	1, 3	1, 3

latency, throughput

slow 64-bit *right* shifts and 64-bit rotations

Rotate shifts on 64-bit Pentium 4

```
rol  rax,1
rol  rbx,1
rol  rcx,1
rol  rdx,1
rol  rsi,1
rol  rdi,1
rol  rbp,1
```

```
rol  rax,1
xor  r9,r9
rol  rbx,1
xor  r9,r9
rol  rcx,1
xor  r9,r9
rol  rdx,1
xor  r9,r9
rol  rsi,1
xor  r9,r9
rol  rdi,1
xor  r9,r9
rol  rbp,1
```

49 cycles (throughput : 1/7) 7 cycles (throughput : 1)

Some Code Examples

	32 bit	64 bit (1)	64 bit (2)

	`xor eax,0[esi+ecx]`
	`xor ebx,4[esi+ecx]`
	`add ecx,8`
		`xor rax,0[rsi+rcx]`
		`xor rbx,8[rsi+rcx]`
		`add rcx,16`
			`xor rax,TABLE+0[rcx]`
			`xor rbx,TABLE+8[rcx]`
			`add rcx,16`

Length	10 bytes	13 bytes	**18 bytes**
Pentium 4	2.2 cycles	2.2 cycles	2.2 cycles
Athlon 64	1.0 cycle	1.0 cycle	**1.4 – 1.9 cycles**

	32 bit	64 bit (1)	64 bit (2)

	`movzx ecx,al`
	`xor ebx,[esi+ecx*4]`
	`shr eax,8`
		`movzx rcx,al`
		`xor rbx,[rsi+rcx*8]`
		`shr rax,8`
			`movzx rcx,al`
			`xor rbx,TABLE[rcx*8]`
			`shr rax,8`

Length	9 bytes	12 bytes	16 bytes
Pentium 4	1.7 cycles	**7.0 cycles**	**7.0 cycles**
Athlon 64	1.0 cycle	1.0 cycle	1.0 cycle

Performance of AES on x64 Processors

- The structure of AES is optimized for 32-bit processors.
- Free from “register starvation” due to 16 general registers.

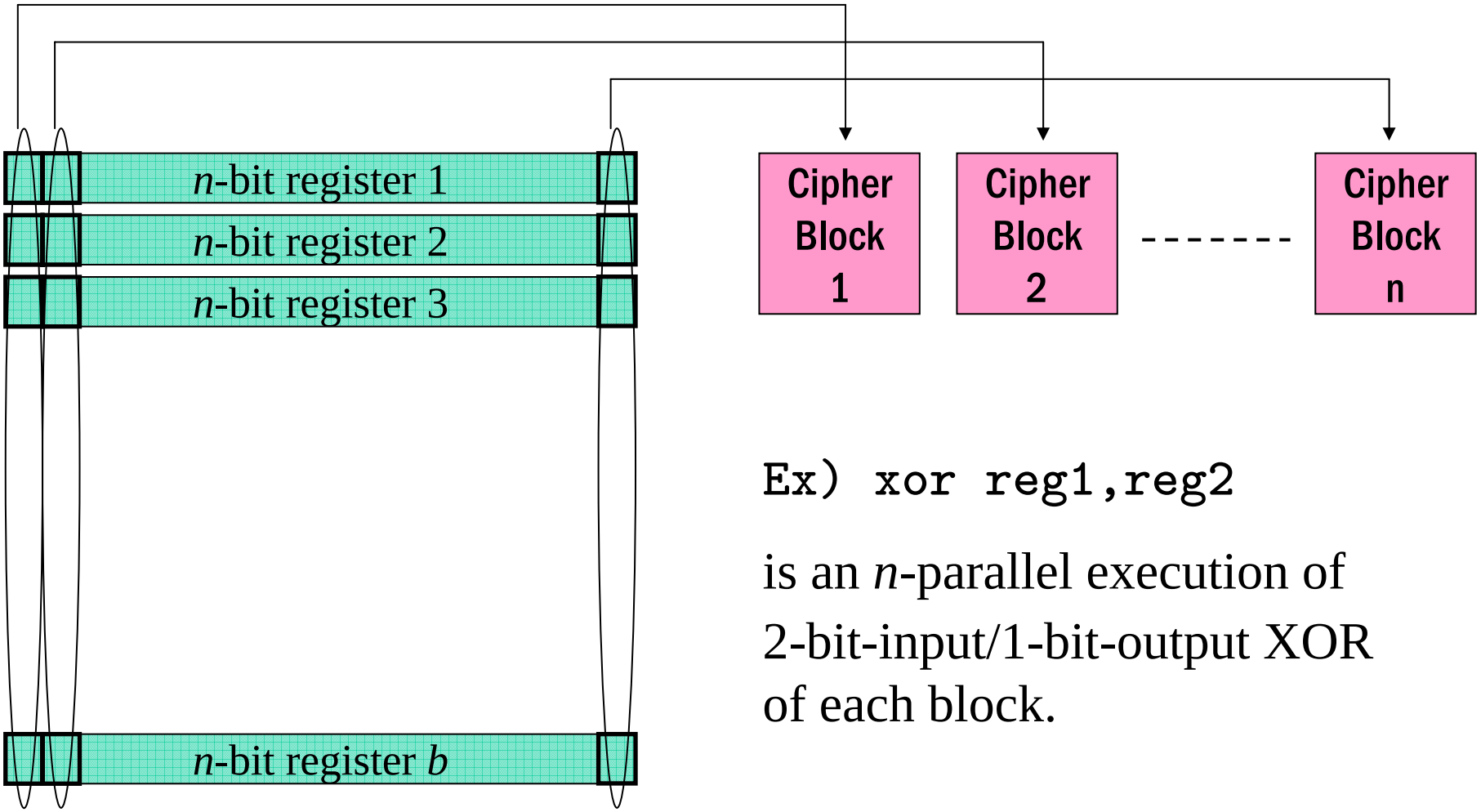
Performance of AES (128-bit key) on Athlon64/Pentium 4

	AES		
Processors	Athlon 64 64-bit	Pentium 4 64-bit	Pentium 4 32-bit
cycles/block	170	256	284
instructions/cycle	2.74	1.81	-
uops/cycle	3.53	2.34	-

Bitslice Implementation of Block Ciphers

- Introduced by Biham (FSE'97)
- *n*-block parallel execution using *n*-bit registers
- *1* software instruction = *n* simple hardware gates
  - `AND, OR, XOR, NOT...`
- Very efficient if
  - registers are long
  - registers are many
  - the target algorithm is small in hardware
- Protection against cache timing attack

Principle of Bitslice Implementation



`Ex) xor reg1,reg2`

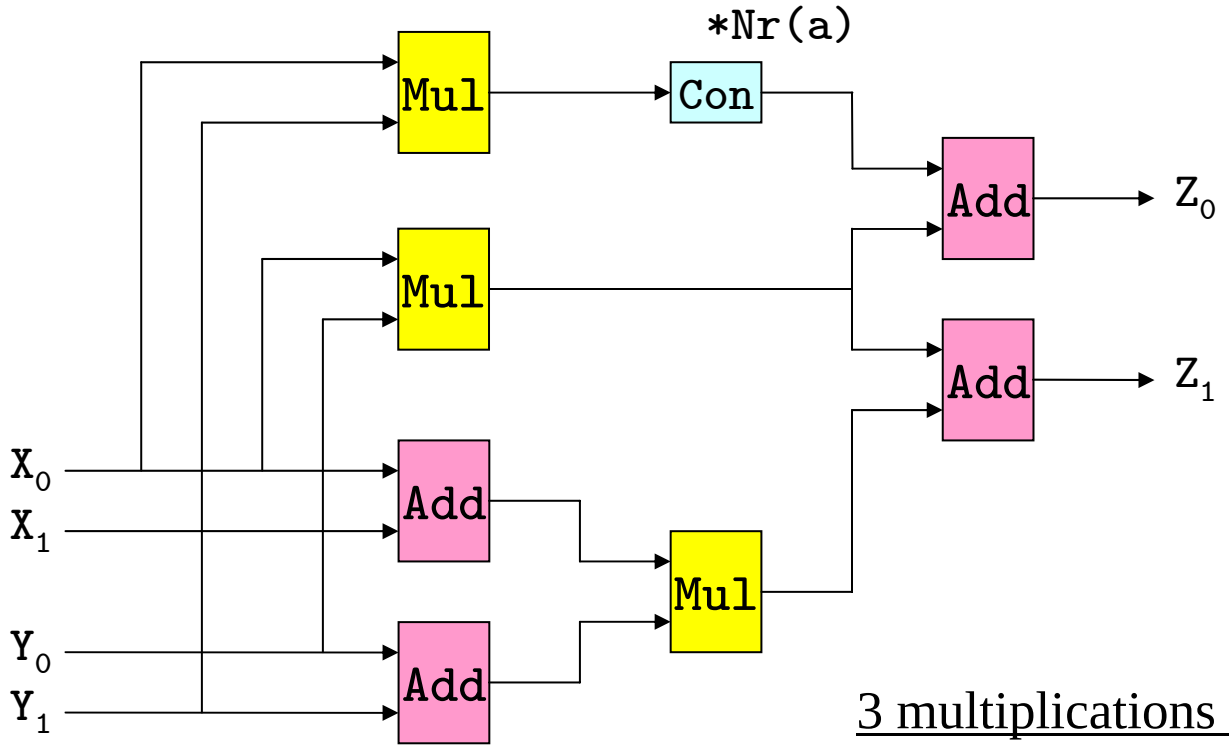
is an *n*-parallel execution of 2-bit-input/1-bit-output XOR of each block.

Bitslice and S-box

- Many recent block ciphers have adopted an 8x8 S-box (a lookup table), linearly equivalent to an inversion over GF(2^8).
  - AES, Camellia, SNOW2.0, ARIA etc
- An inversion over GF(2^8) is strong against differential/linear attacks (actually best known), but can be weak against cache timing attacks.
- The bitslice implementation can compute an inversion over GF(2^8) without a table lookup.

Multiplication over GF(2^{2n}) using GF(2^n)

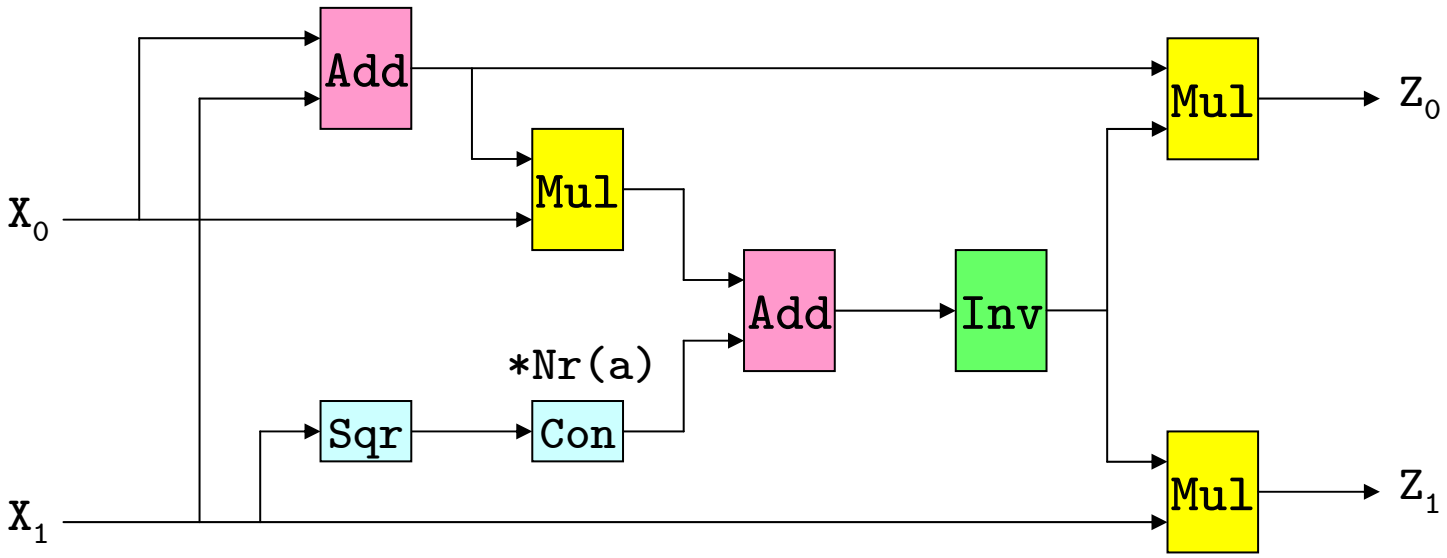
Basis of GF(2^{2n})/GF(2^n): (1, a)



3 multiplications over GF(2^n)

$Z_0+Z_1a = (X_0+X_1a)(Y_0+Y_1a)$ where $Tr(a)=1$

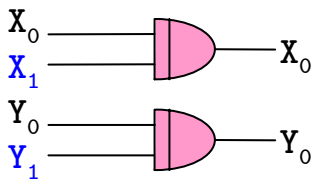
Inversion over GF(2^{2n}) using GF(2^n)



$Z_0+Z_1a = 1/(X_0+X_1a)$ where $Tr(a)=1$

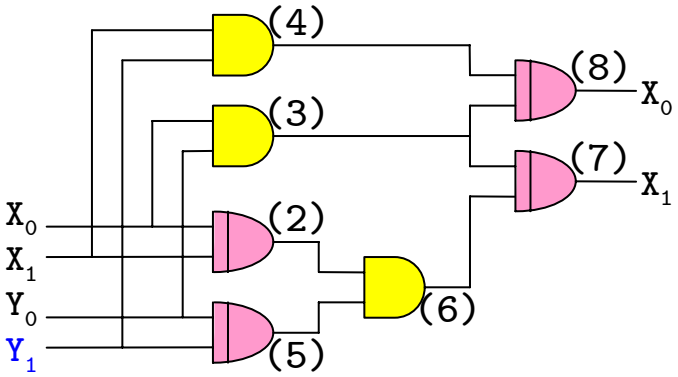
Circuits on GF(2^2)

Addition



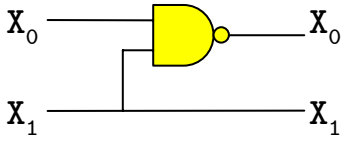

```
xor x0,x1
xor y0,y1
```

Multiplication




```
(1)  mov t0,x1 ; t0 temporary
(2)  xor x1,x0
(3)  and x0,y0
(4)  and t0,y1
(5)  xor y0,y1
(6)  and x1,y0
(7)  xor x1,x0
(8)  xor x0,t0 ; y1 unchanged
```

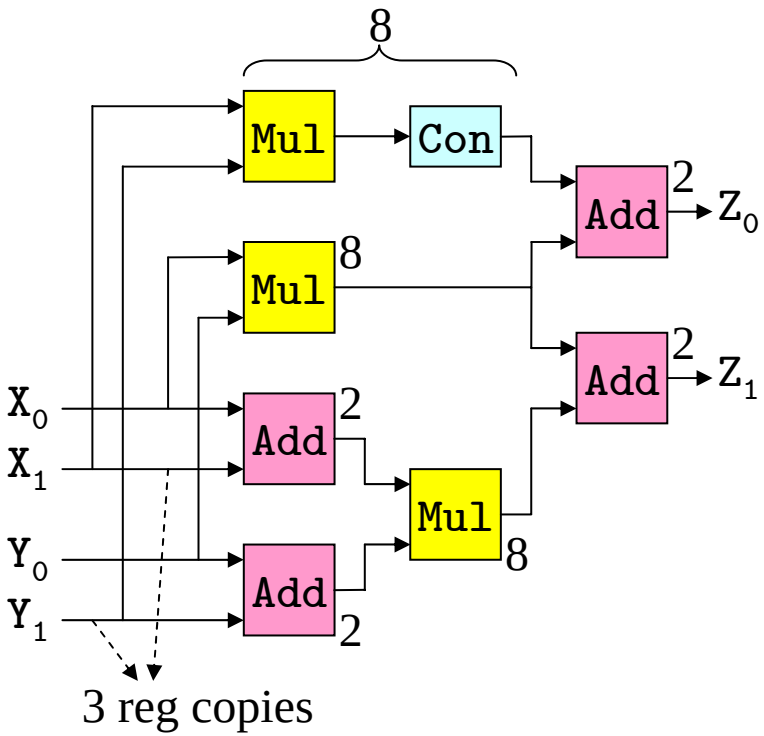
Inversion



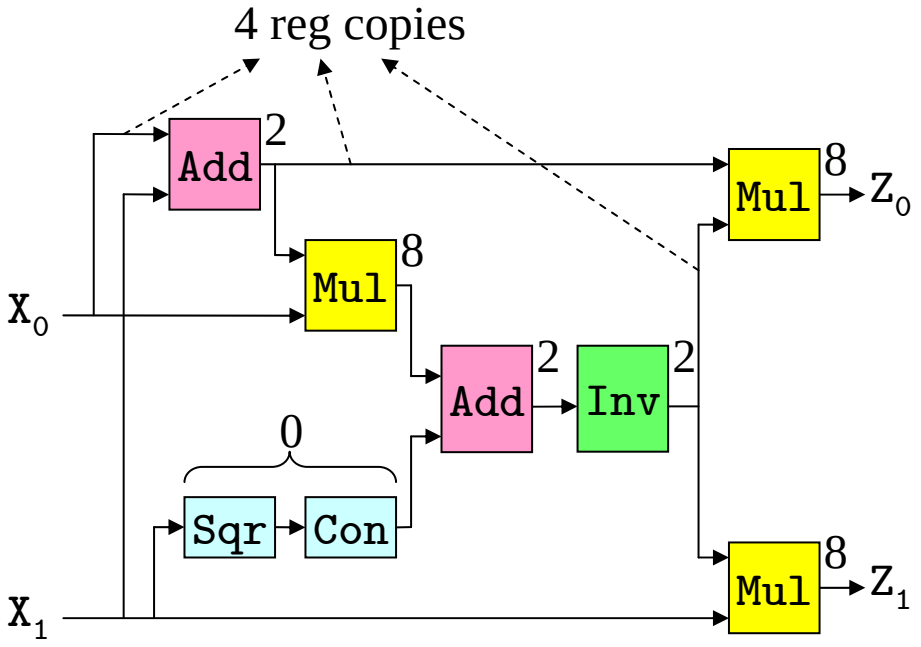

```
and x0,x1
not x0
```

Multiplication/Inversion on GF(2^4)

Multiplication



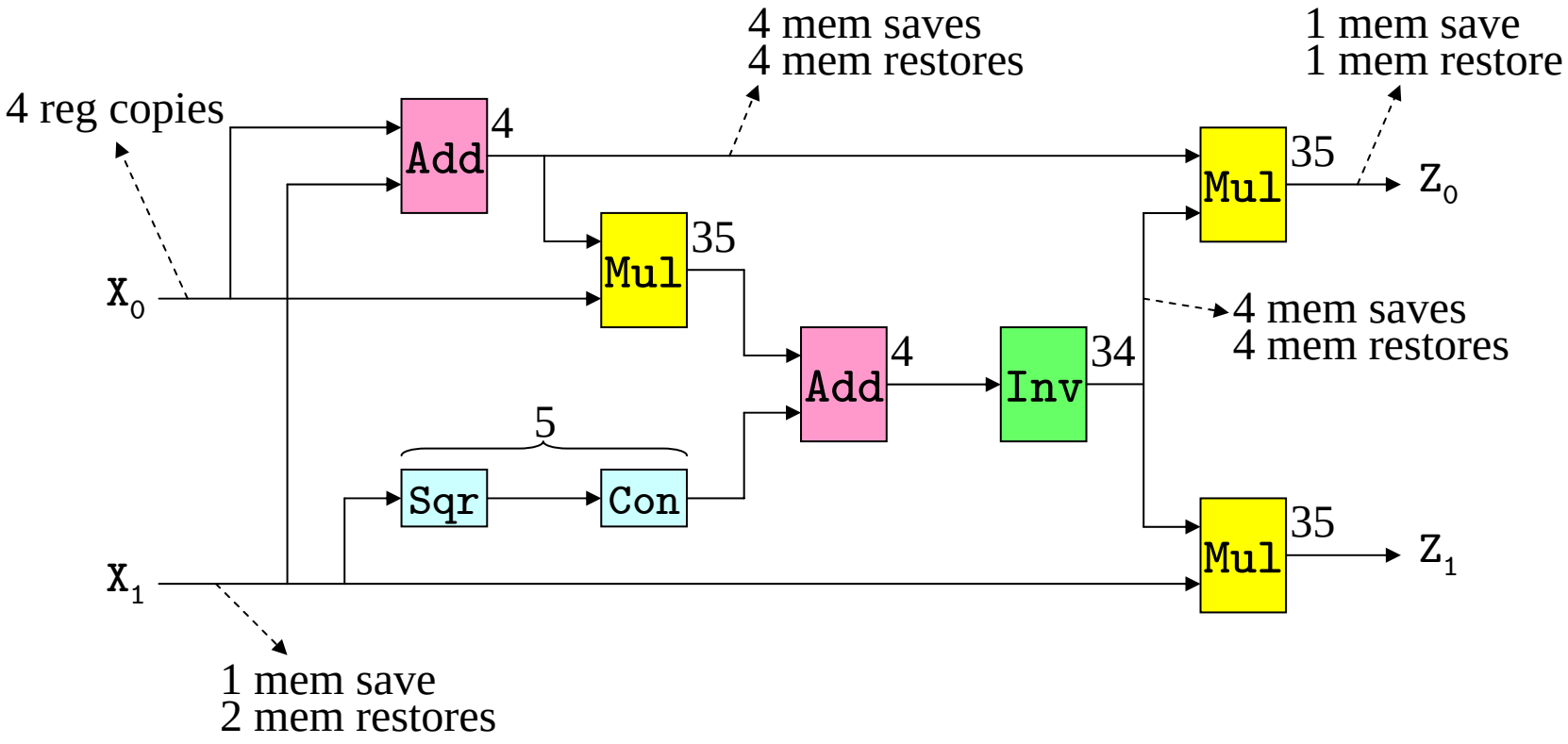
Inversion



35 instructions with 3 temp. registers

34 instructions with 4 temp. registers

Inversion on GF(2^8)



177 instructions (156 reg-reg's + 21 mem-reg's)

Implementation Results

The Full AES S-box

Basis Change (before inversion)	Inversion	Basis Change (after inversion)	Constant XOR	Total
12	177	16	(4)	205 (209)

Performance of Bitsliced AES/Camellia on Athlon64/Pentium 4

	AES		Camellia	
Processors	Athlon 64	Pentium 4	Athlon 64	Pentium 4
cycles/block	250	418	243	415
instructions/cycle	2.75	1.66	2.74	1.61
uops/cycle	3.20	1.93	2.99	1.75

Concluding Remarks

- A combination of lookup tables and logical operations is suitable for both software and hardware.
- Understanding hardware is important in doing software.
- Pentium 4 looks a dead end of processor design
  - The long pipeline leads to an overheating problem
  - AMD Athlon64 very often runs faster than Pentium 4
- Parallel encryption will be increasingly important
- Intel's new 'Core' processors go back to Pentium III
  - Bitsliced ciphers can be much faster on Core2