Fundusze Europejskie
Wiedza Edukacja Rozwój

Rzeczpospolita Polska

Unia Europejska
Europejski Fundusz Społeczny

Politechnika Śląska jako Centrum Nowoczesnego Kształcenia
opartego o badania i innowacje

POWR.03.05.00-IP.08-00-PZ1/17

Projekt współfinansowany przez Unię Europejską ze środków Europejskiego Funduszu Społecznego

Microprocessor and Embedded Systems

Faculty of Automatic Control, Electronics and Computer Science,
Informatics, Bachelor Degree

Lecture 14

64-bit microprocessors
Intel Itanium

Bartłomiej Zieliński, PhD, DSc

Intel Itanium

Program:

- Main assumptions
- EPIC architecture basics
- Registers, execution units
- Instruction format
- Jump predicAtion

Intel Itanium

- Main assumptions
  - New approach
    - Command-level parallelism support
      - Different from superscalar architecture
    - Predicative execution
      - Each command is related to a 1-b predicate
      - Command executed, when predicate=1

      → speculative branch execution and conditional instr.

      → result returned when condition is determined

Intel Itanium

- Main assumpions
  - New approach
    - Control speculation
      - Load operations moved earlier
      - Exception → served only if the load was really necessary
    - Data speculation
      - Load before write command to the read cell
      - Then check before use if value still valid
    - Software pipeline
      - Parallel loop iterations execution

Intel Itanium

- EPIC architecture
  - *Explicit Parallel Instruction Computer*
    - Command-level parallelism visible in the code
      - Not need to determine it during execution
    - Very long instruction words (→ VLIW architecture)
    - Branch predicAtion
    - Speculative load

Intel Itanium

- EPIC architecture
  - Explicit Parallel Instruction Computer
    - Reasonable utilisation of many million transistors
      - Not being cache
      - Without increasing superscalarity level
    - Commands scheduled statically by the compiler
      - Not dynamically by the µp
      - Parallel execution possibility set by the compiler and used by the µp during execution
      - µp structure simplified
      - Better optimisation
        - » compiler has more resources to optimise code well

Intel Itanium

- EPIC architecture
  - Many registers
    - 128×64-b integer
    - 128×80-b floating point
    - 64×1 predication

    → high parallelism level
  - Many execution units
    - More than 8
      - (superscalar → abt. 4)

Intel Itanium

- EPIC architecture
  - Registers
    - Superscalar
      - Few visible registers, many aliases (reg. rename)
    - EPIC
      - Many visible registers (explicit parallelism)
  - Execution units
    - Depend on available number of transistors
      - E.g., if 8 commands to be performed in paralel, but only 4 exec.units, execution takes 2 cycles
      - If 8 commands and 8 units → 1 cycle

Intel Itanium

- IA-64
  - 4 execution units types
    - I(nteger)
      - arithmetical/logical, shift+add, comp, integer multimedia
    - M(emory)
      - reg↔mem transfers, some integer ALU
    - B(ranch)
      - Jumps, branches
    - F(loating)
      - FPU operations

Intel Itanium

- IA-64
  - 6 command types

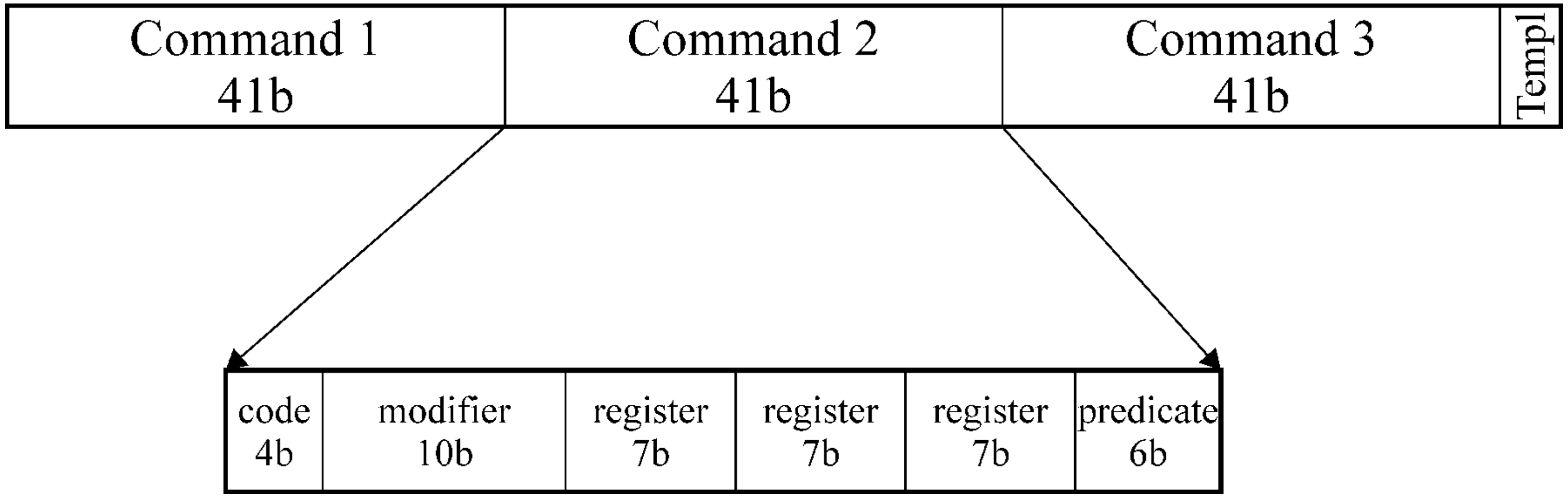
Symbol	Type	Explanation	Exec.unit
A	ALU	Integer ALU	I/M
I	Integer	Integer outside of ALU	I
M	Memory	Memory operations	M
F	FPU	Floting point	F
B	Branch	Jumps	B
L+X	Extended	extended	I/B

Intel Itanium

- Itanium – command format
  - 128-b bundle = 3 commands + template
  - µp fetches 1 or more bundles
  - Template
    - Which commands to be executed in paralel
    - Group can extend to multiple bundles
  - µp browses many bundles
    - E.g., 8 commands paralel
      - Compiler puts them to neighbouring bundles
      - Sets template value

Intel Itanium

- Itanium – command format
  - Commands in bundle – free sequence
    - May differ from source code sequence
  - Template splits bundle into dependent & independent commands
  - No need to fill bundles with NOPs (opp. to VLIW)



Intel Itanium

- Itanium – command format
  - Main operation code interpretation
    - Template-dependent
    - Bundle-location-dependent
  - 41b length
    - > 32b in RISC
    - < 118b µop Pentium Pro
    - Results from:
      - Many registers
      - Predicate register

Intel Itanium

- Itanium – command format
  - Some example templates

Template	Slot1	Slot2	Slot3
00, 01	M	I	I / I;;
02, 03	M	I;;	I / I;;
04, 05	M	L	X / X;;
08, 09	M	M	I / I;;

  - Assembler command

```
[qp] mnem [.compl]  dst=srcs [;;]
```

Intel Itanium

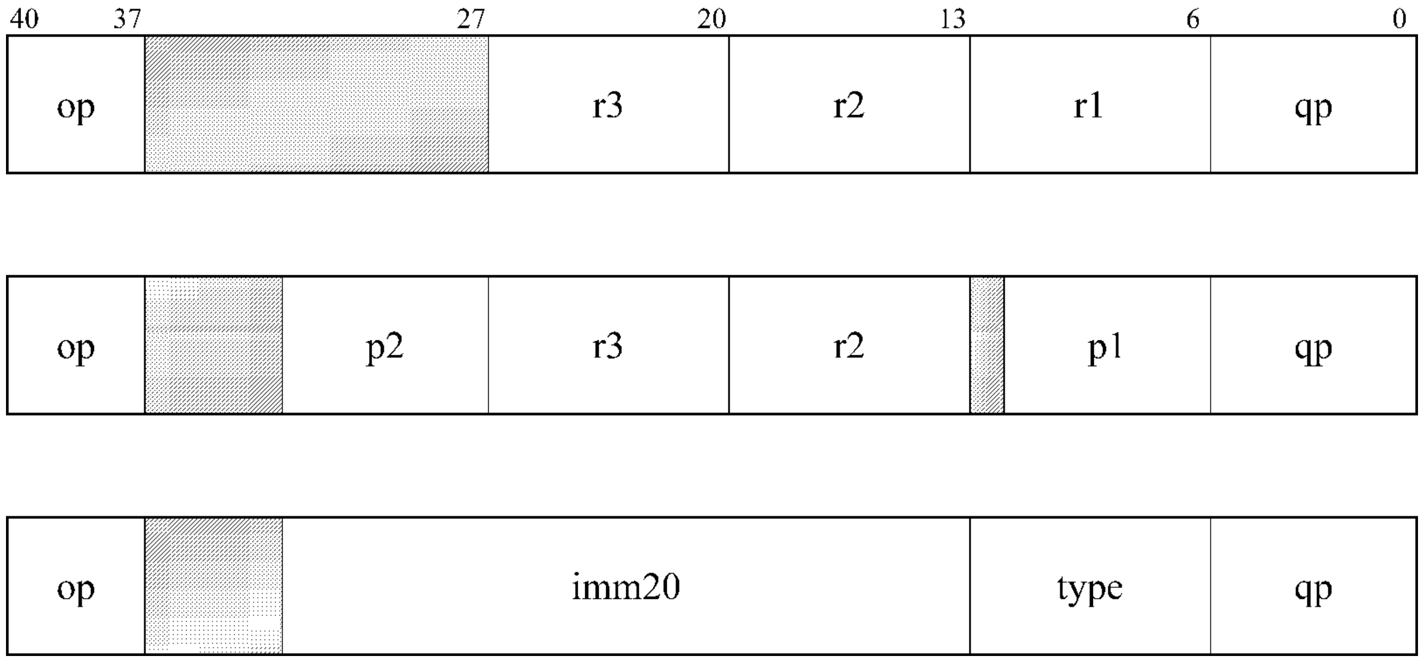
- Itanium – command format
  - Assembler command

```
[qp] mnem [.compl]  dst=srcs [;;]
```

- Qp – qualifying predicate
  - QP0 – independent command
- Mnem – command name
- Compl. – command completer
- Dst – one or more destination argument(s)
- Srcs – source arguments (typically 2)
- ;; – group borders

Intel Itanium

- Itanium – command format
  - Few example formats
    - Arithmetical/logical
    - Compare
    - Jumps



Intel Itanium

- Itanium – command format
  - Few example commands

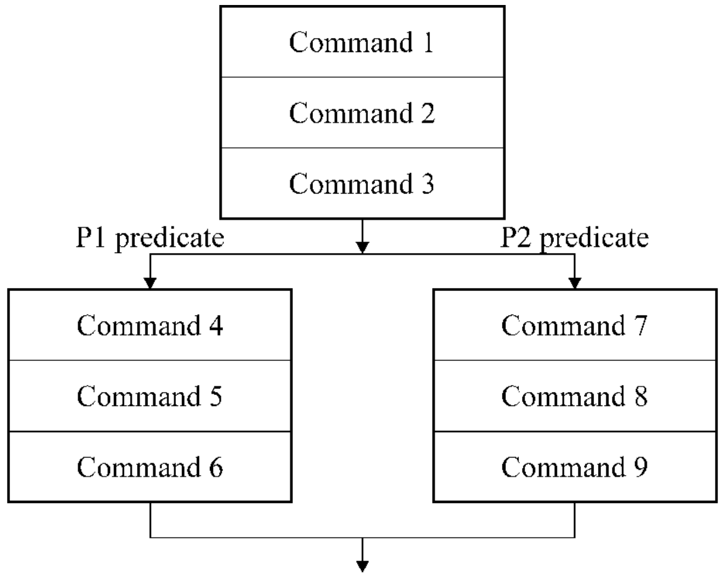
Command	Operation	Comment
Add r5=r6,r7	R5←r6+r7	Addition
Ld4 r2=[r3]	$R2_{31..0}$←M[r3]	Load 4B reg→mem
(p4) add r1=r2,r3	P4=>r1←r2+r3	Add when p4=1; else nop
(p6) add r8=-1,r32	P6=>r8←r32-1	Add const when p6=1; else nop
Cmp.eq p6,p5=r32,r0	P6←(r32=r0); p5←not p6	Check if equal; set p5,p6
Extr.u r31=r2,3,6	$R31_{5..0}$←$r2_{8..3}$	Extract selected bits to another reg
Cmp.eq.and p1,p2=r32,r33	(r32≠r33)=>p1=p2=0	Check if equal

Intel Itanium

- Itanium – predication
  - Compiler determines instructions to be executed in parallel
  - Branches replaced with conditional execution
    - No branch predicion necessary

Intel Itanium

- Itanium – predication
  - 2 paths → 2 PRs
  - Parallel path execution
    - No dependencies between paths
  - Comparison result known → useless result removed
  - Compiler can change commands sequence for parallel execution



Command 1	Command 2	Command 3
Command 4	Command 7	Command 5
Command 8	Command 6	Command 9

Intel Itanium

- Itanium – predication
  - Example 1

```
if (r1==r2)
  r3 = r4+r5
else
  r6 = r4-r5;
```

```
        Cmp r1, r2
        Jne NotEq
        Mov r3, r4
        Add r3, r5
        Jmp Further
NotEq:
        Mov r6, r4
        Sub r6, r5
Further:
```

```
     cmp.eq p1, p2=r1, r2
(p1) add r3=r4, r5
(p2) sub r6=r4, r5
```

Branch prediction
Risk of misprediction

Parallel execution of all cmds
Only proper result accepted

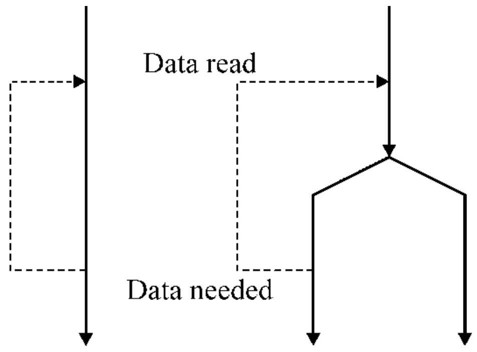
Intel Itanium

- Itanium – predication
  - Example 2

```
If (a && b)                  cmp.eq p1,p2=0,a      // a=0?
  j++;                  (p2) cmp.eq pq, p3=0,b     // b=0?
Else                    (p3) add j=1,j             // j++
  if (c)                (p1) cmp.ne p4,p5=0,c      // c≠0?
    k++;                (p4) add k=1,k             // k++
  else                  (p5) add k=-1,k            // k--
    k--;                     add i=1,i             // i++
i++;
```

Intel Itanium

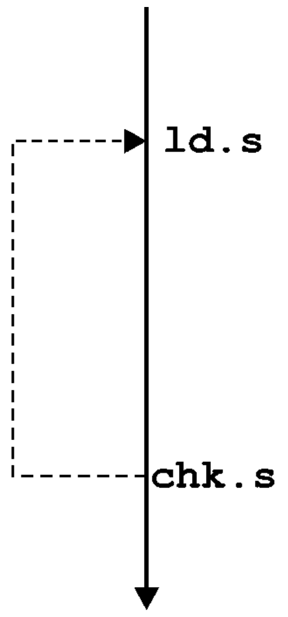
- Itanium – speculative load
  - Data read from mem earlier than needed in µp
    - No delay, no wait until read finished
  - Load operation moved earlier in code



  - *But what if load moved before cond. branch?*
    - E.g., predicative load: mem read, but reg filled only when predicate value known
    - Exception possible (address error, paging error, etc.)

Intel Itanium

- Itanium – speculative load
  - In IA-64, load separate from exception
    - Ld.s (speculative load): mem read, exception detected but not signalled
    - Chk.s (check speculative): in original load location; can be predicated; signals exception if necessary



    - ld.s exception detected → NaT („*Not a Thing*”) bit set for a destination reg.
    - Chk.s checks NaT; if =1, exception signalled

Intel Itanium

- Itanium – speculative load
  - Example

```
Int funct (int *x) {
  if (x==NULL)
    return -1;
  else
    return (*x+7);
} /* funct */
```

```
Funct:
  ld8.s r1=[r32]
  cmp.eq
p6,p5=r32,r0;;
(p6) add r8 -1,r0
(p6) br.ret
(p5) chk.s r1,err;;
  add r8=7,r1
  br.ret
………………
Err:
  ld8 r1=[r32]
  add r8=7,r1
  br.ret
```

Intel Itanium

- Itanium – advanced load
  - Load before read location written
  - Check if read data still valid

  - Load writes address to ALAT
    - *Advanced Load Address Table*
  - Write checks ALAT
    - if addr match, address removed from ALAT
  - Check checks ALAT
    - Addr found → data valid
    - Addr not found → data invalid, load repeat

Intel Itanium

- Itanium – speculative load
  - Example

```
Add r3=4,r0;;
St4 [r32]=r3
Ld4 r2=[r33];;
Add r5=r2,r3
```

```
Ld4.a r2=[r33]
Add r3=4,r0;;
St4 [r32]=r3
Ld4.c r2=[r33];;
Add r5=r2,r3
```

```
Ld8.a r6=[r8]// advanced load; ALAT write
St8 [r4]=r12 // write and ALAT check
Ld8.c r6=[r8]// load check; ALAT check
```

Intel Itanium

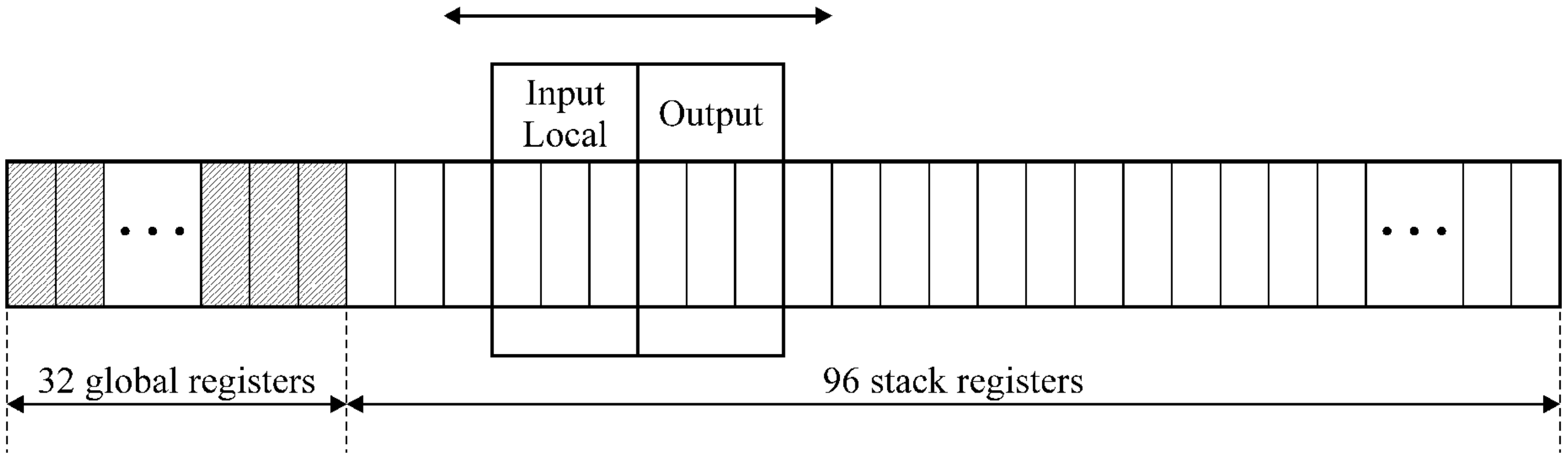
- Software pipelining
  - Parallel different loop iteration execution
  - Support for this technique:
    - Auto register rename
      - Different reg. subsets used for different iterations
    - Predication
      - Rotating predicate reg specifies execution phase
        - » prolog, core, epilog
    - Specialised loop end command
      - Reg rotation
      - Loop count decrement

Intel Itanium

- Registers
  - 128×64-b integer
    - R0..R31 static
    - R32..R127 dynamic/rotating
    - NaT bit for each reg
  - 128×80-b floating point
    - IEE 754 double extended format
    - R0..R31 static
    - R32..R127 dynamic/rotating
  - 64×1 predication
    - Pr0=1 → unconditional commands
    - Pr0..15 static
    - pr16..63 rotating

Intel Itanium

- Register stack
  - Prevents unnecessary data move mem↔reg when calling a subroutine
  - Auto frame assignment
    - up to 96 regs for a procedure



Intel Itanium

- Register stack
  - Procedure call
    - Hide local regs of calling procedure
    - Auto register rename
    - Cyclic-buffered assignment
      - If not enough registers, oldest stored in mem