Computer Architecture
(Addressing Modes in ISA)

Subhasis Bhattacharjee

Department of Computer Science and Engineering,
Indian Institute of Technology, Jammu

May 28, 2023

What is Addressing Mode in an ISA?

An addressing mode is the method of specifying and accessing an operand in an assembly language statement.

- It defines "how the machine language instructions identify the operand(s) of each instruction".
- Here, "operand" is not restricted to source / inputs of an instruction
- It also specifies the location and writing of result after operation is performed.
- It specifies "how to calculate the effective memory address of an operand" by using:
  - information held in registers, and/or
  - constants contained within a machine instruction or elsewhere.
- An ISA can support multiple addressing modes

Instruction Classification based on number of operands

If an instruction takes n operands, then it is said to be in the n-address format.

Examples: (Here AC implies the accumulator register)

- **add r1** ($AC \leftarrow AC + r1$). It is 1 address format.
- **add r1, r2** ($r1 \leftarrow r1 + r2$). It is 2 address format.
- **add r1, r2, r3** ($r1 \leftarrow r2 + r3$). It is 3 address format.
- Other address formats are also possible.

Various Addressing Modes

- Immediate
- Direct
- Indirect
- Register
- Register Indirect
- Displacement
  - Indexed
  - Base-Index
  - Base-Index-Offset
- Stack

Immediate Addressing Mode

- Operand is part of instruction
- **add r1, #5** ($r1 \leftarrow r1 + 5$)
- **mov r2, #34** ($r2 \leftarrow 34$)

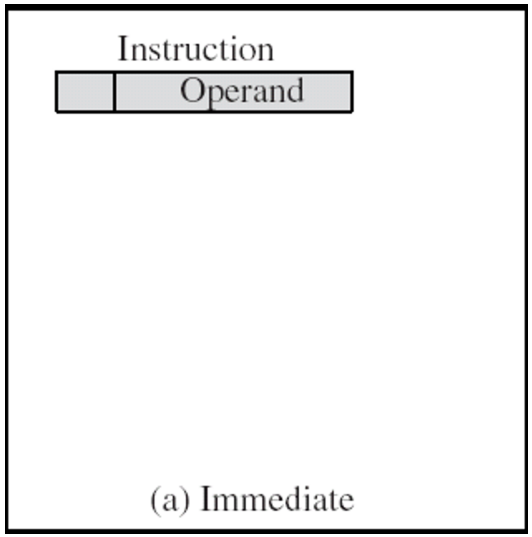
Advantages:

- No memory reference to fetch data
- Fast

Drawbacks:

- Limited range of values - limited by the number of bits available for this operand in the instruction format.
  - Eg, If only 12 bits are available in the instruction format - then values can range from 0 to $2^{12} - 1$ (for unsigned numbers).
  - The range of values will be even smaller for signed numbers.

Immediate Addressing Mode Diagram



Some misnomer

- Addressing mode does not always mean memory address
- It can specify immediate operand value - as discussed in previously
- It can indicate a CPU register which contains the operand

Direct Addressing

- Address field contains address of operand.
- Effective address EA = address field (A)

data: .word 0x1234
add r1, data ($r1 \leftarrow r1 + 0x1234$)

- Look in memory at address **data** for operand
- Fetch the operand value 0x1234 from that

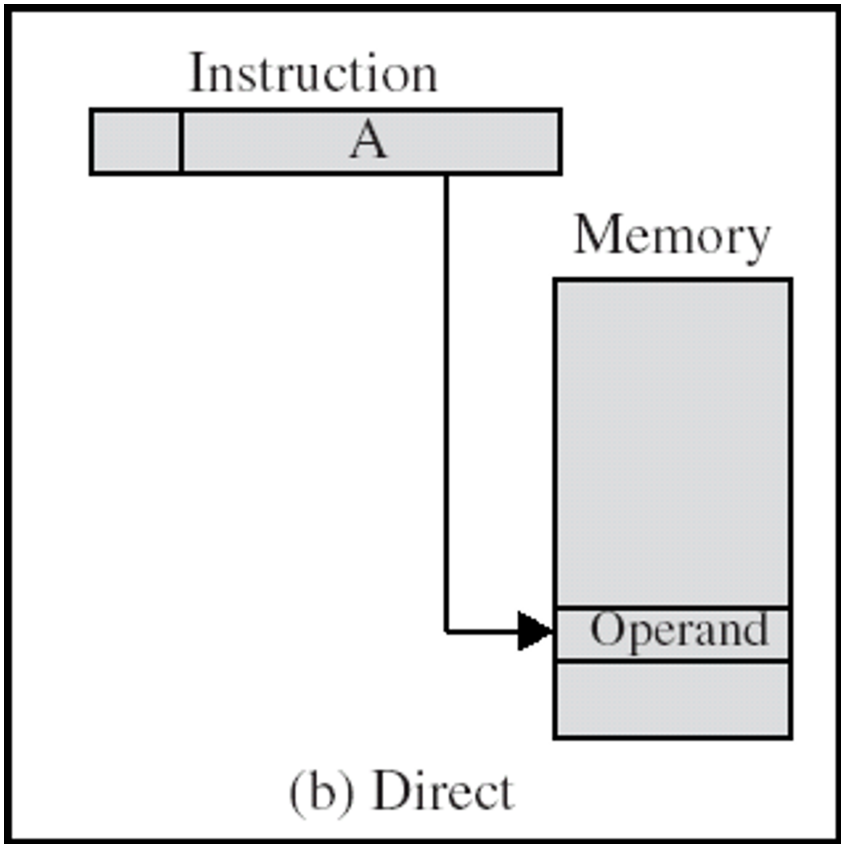
Advantages:

- Single memory reference to access data.
- No additional calculations to work out effective address.

Drawbacks:

- Limited address space - limited by the number of bits available for this operand in the instruction format.

Direct Addressing Mode Diagram



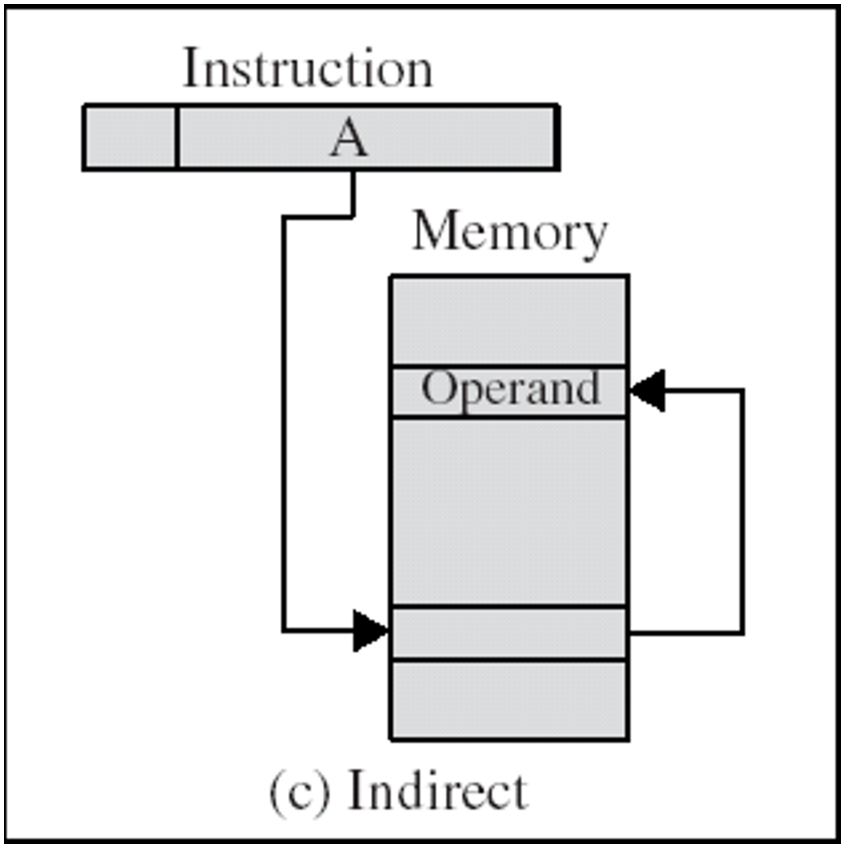
Indirect Addressing (1/2)

- Memory cell pointed to by address field contains the address of (pointer to) the operand
- EA =(A)
  - Look in A, find address (A) and look there for operand
- e.g. ADD AX, (A)
  - Add contents of cell pointed to by contents of A to accumulator

Indirect Addressing (2/2)

- Large address space
- 2n where n = word length
- May be nested, multilevel, cascaded
  - e.g. EA = (((A)))
    Draw the diagram yourself
- Multiple memory accesses to find operand
- Hence slower7

Indirect Addressing Diagram



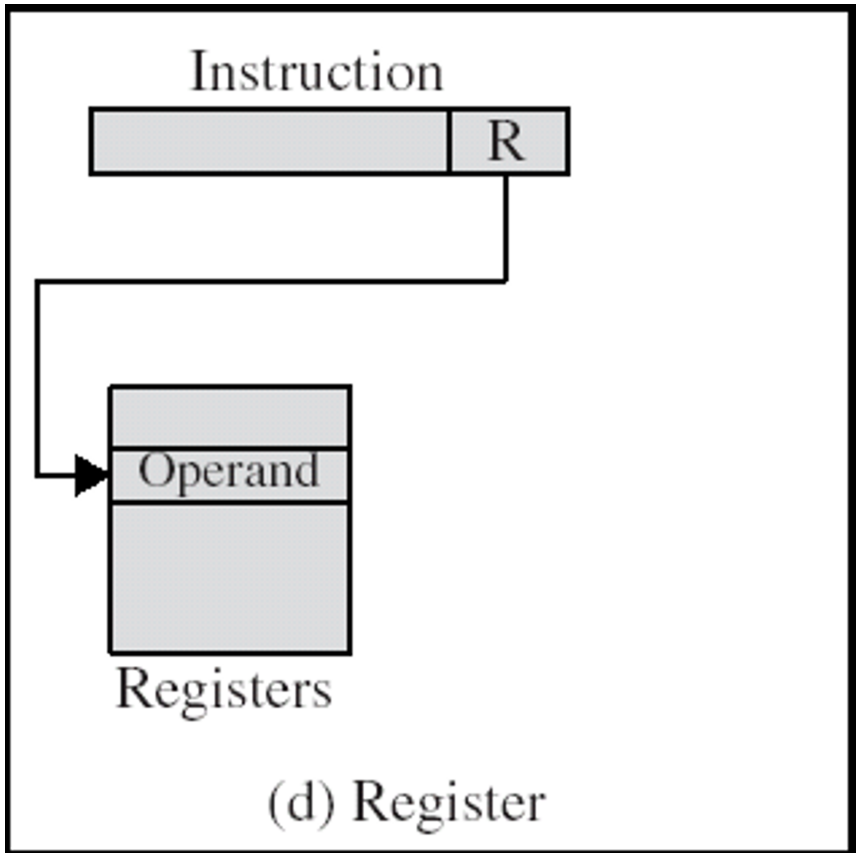
Register Addressing (1/2)

- Operand is held in register named in address filed
- EA = R
- Limited number of registers
- Very small address field needed
  - Shorter instructions
  - Faster instruction fetch
  - MOV AX, BX
  - ADD AX, BX9

Register Addressing (2/2)

- No memory access
- Very fast execution
- Very limited address space
- Multiple registers helps performance
  - Requires good assembly programming or compiler writing
  - N.B. C programming
    register int a;
  - c.f. Direct addressing

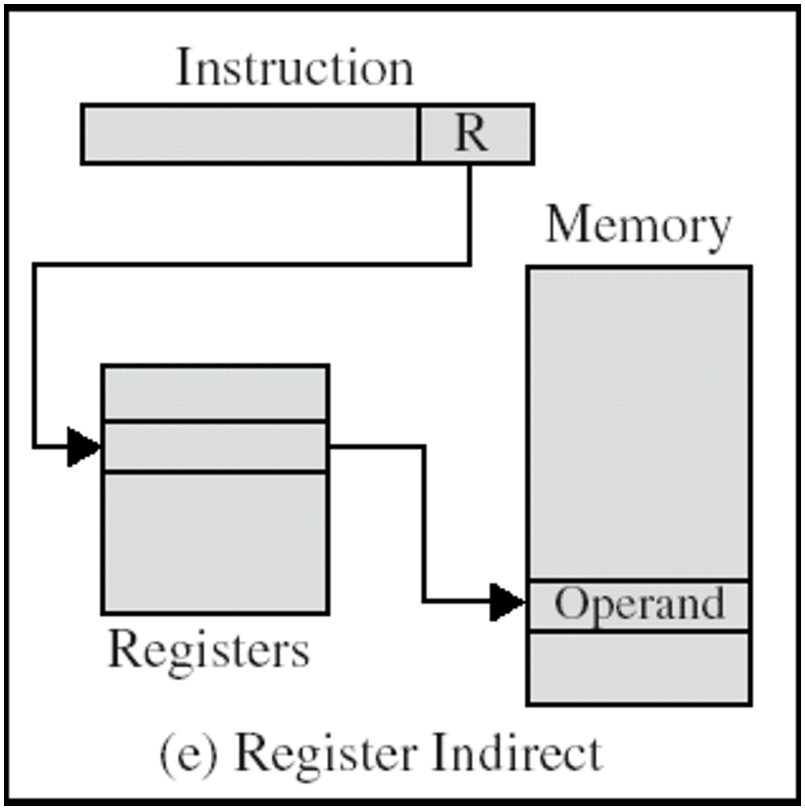
Register Addressing Diagram



Register Indirect Addressing

- C.f. indirect addressing
- EA = (R)
- Operand is in memory cell pointed to by contents of register R
- Large address space (2n)
- One fewer memory access than indirect addressing12

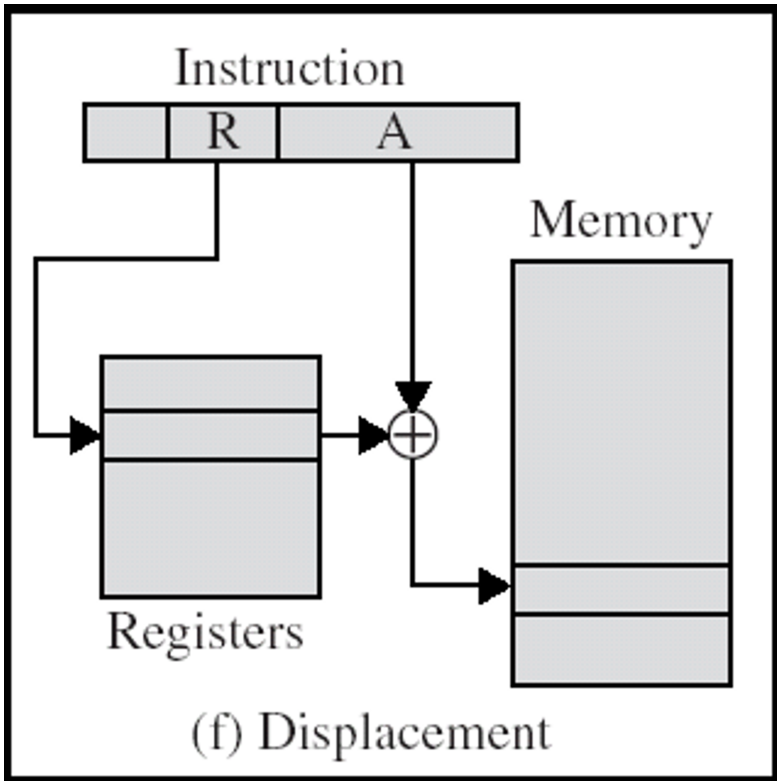
Register Indirect Addressing Diagram



Displacement Addressing

- EA = A + (R)
- Effective address=start address + displacement
- Effective address=Offset + (Segment Register)
- Use direct and register indirect
- Address field hold two values
  - A = base value
  - R = register that holds displacement
  - or vice versa
  - Starting address of > 4 bits 14

Displacement Addressing Diagram



Register Transfer Notation

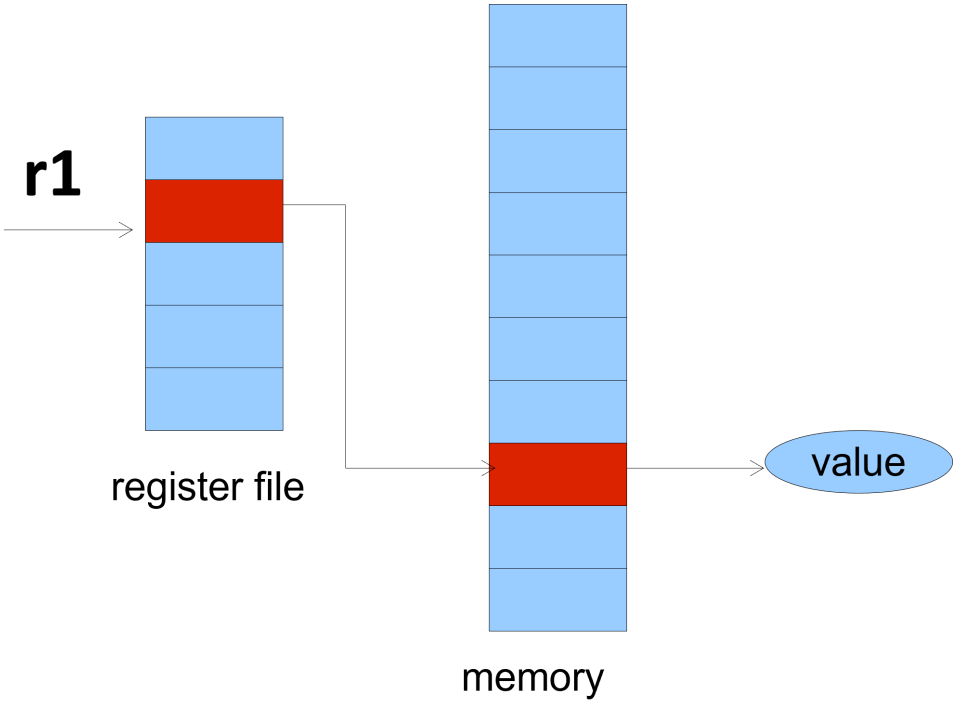
- This notation allows us to specify the semantics of instructions
- r1 $\leftarrow$r2
  - transfer the contents of register r2 to register r1
- r1 $\leftarrow$r2 + 4
  - add 4 to the contents of register r2, and transfer the contents to register r1
- r1 $\leftarrow$[r2]
  - access the memory location that matches the contents of r2, and store the data in register r117

Addressing Modes

- Let V be the value of an operand, and let r1, r2 specify registers
- Immediate addressing mode
  - V ←imm , e.g. 4, 8, 0x13, -3
- Register direct addressing mode
  - V ←r1
  - e.g. r1, r2, r3 . . .
- Register indirect
  - V ←[r1]
- Base-offset : V ←[r1 + offset], e.g. 20[r1] (V ←[20+r1])18

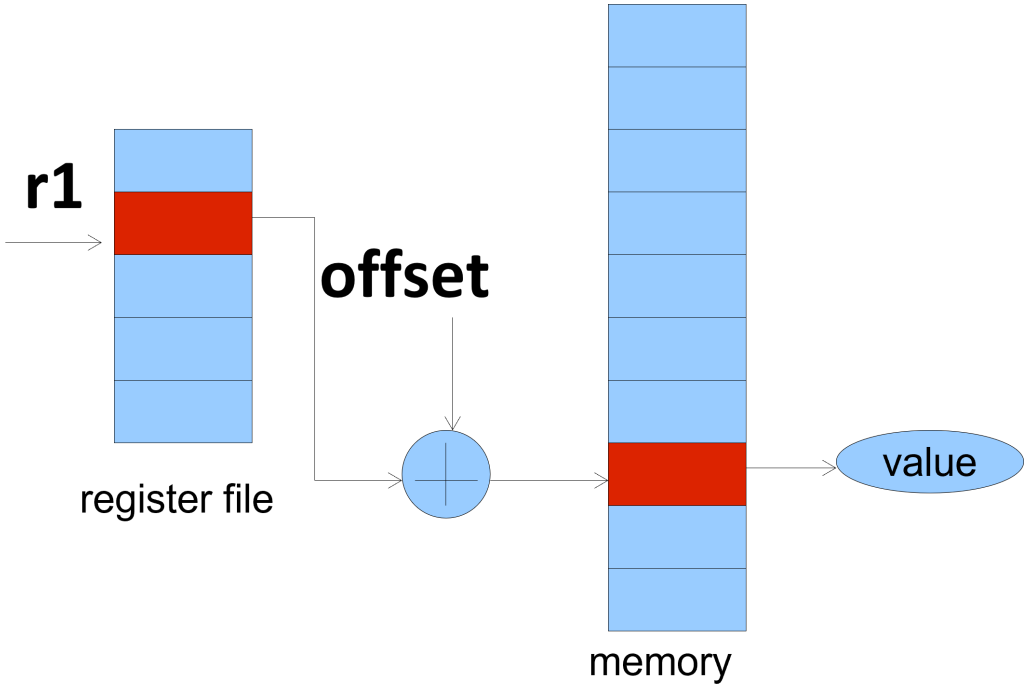
Register Indirect Mode

V ←[r1]



Base-offset Addressing Mode

V ←[r1 + offset]

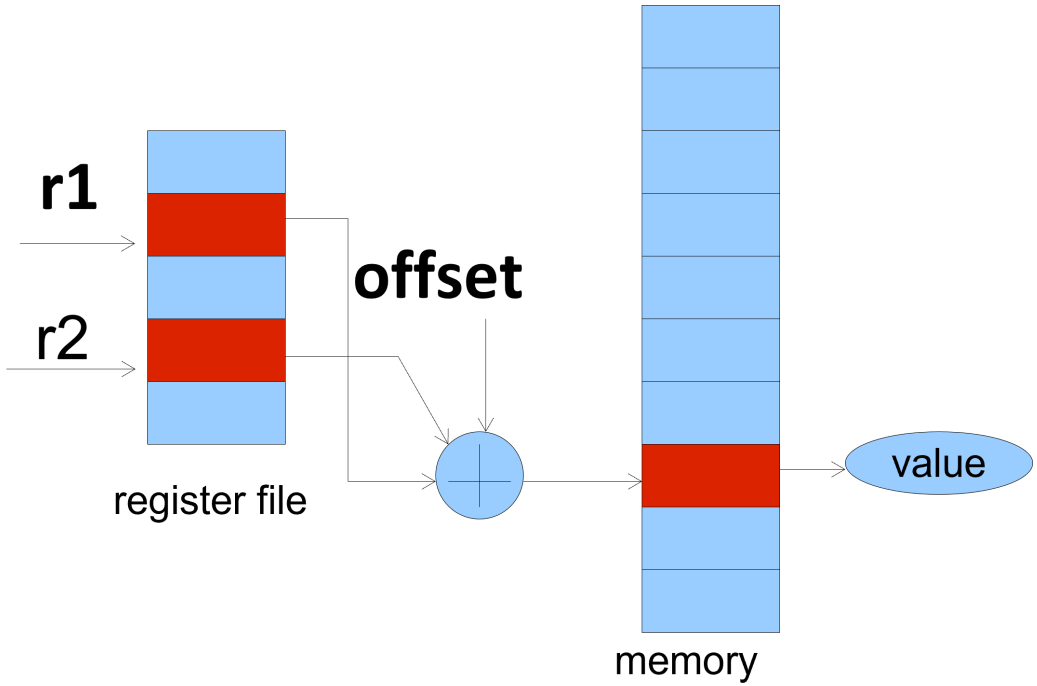


Addressing Modes - II

- Base-index-offset
  - V ←[r1 + r2 + offset]
  - example: 100[r1,r2] (V ←[r1 + r2 + 100])
- Memory Direct
  - V ←[addr]
  - example : [0x12ABCD03]
- PC Relative
  - V ←[pc + offset]
  - example: 100[pc] (V ← [pc + 100])21

Base-Index-Offset Addressing Mode

V $\leftarrow$[r1 + r2 + offset]



Relative Addressing (PC-Relative)

- A version of displacement addressing
- R = Program counter, PC
- EA = A + (PC)
- i.e. get operand from A cells from current location pointed to by PC
- c.f locality of reference & cache usage .text

Relative Addressing (PC-Relative) - Example - TODO

Base-Register Addressing

- A holds displacement
- EA = (CS) + A
- R holds pointer to base address
- R may be explicit or implicit
- e.g. segment registers in 80x8624

Indexed Addressing

- A = base
- R = displacement
- EA = A + (R)
- Good for accessing arrays
- EA = A + (R)
- R++25

Combinations

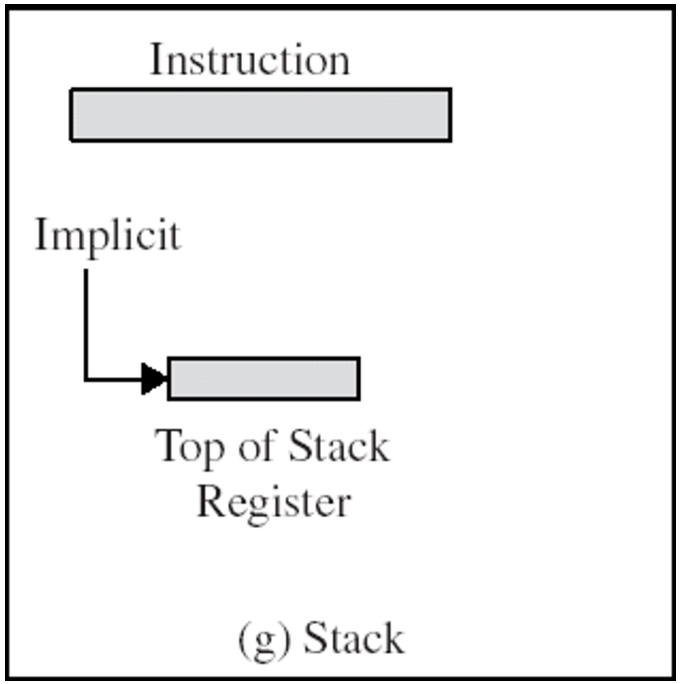
- Postindex
- EA = (A) + (R)
- Preindex
- EA = (A+(R))26

Summary - TODO

Table

Stack Addressing

- Operand is (implicitly) on top of stack
- e.g.
- ADD Pop top two items from stack and add and push



Pentium Addressing Modes

- Virtual or effective address is offset into segment
  - Starting address plus offset gives linear address
  - This goes through page translation if paging enabled
  - 9 addressing modes available
  - Immediate
  - Register operand
  - Displacement
  - Base
  - Base with displacement
  - Scaled index with displacement
  - Base with index and displacement
  - Base scaled index with displacement
  - Relative

Pentium Addressing Mode Calculation

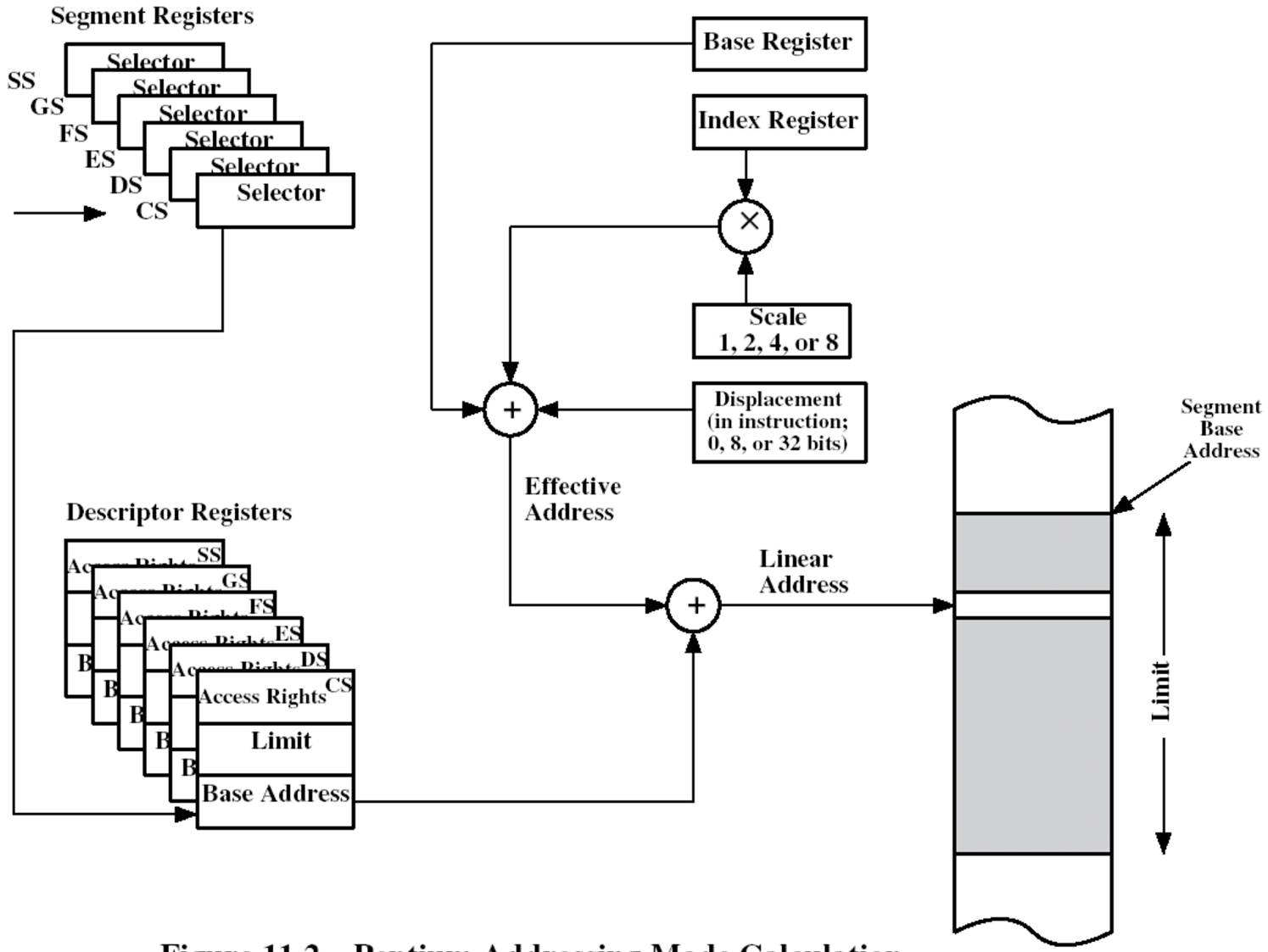
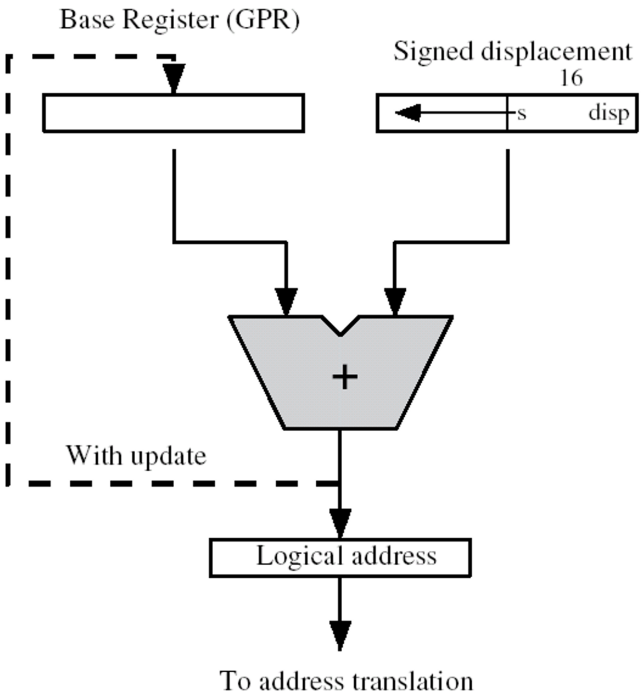


Figure 11.2 Pentium Addressing Mode Calculation

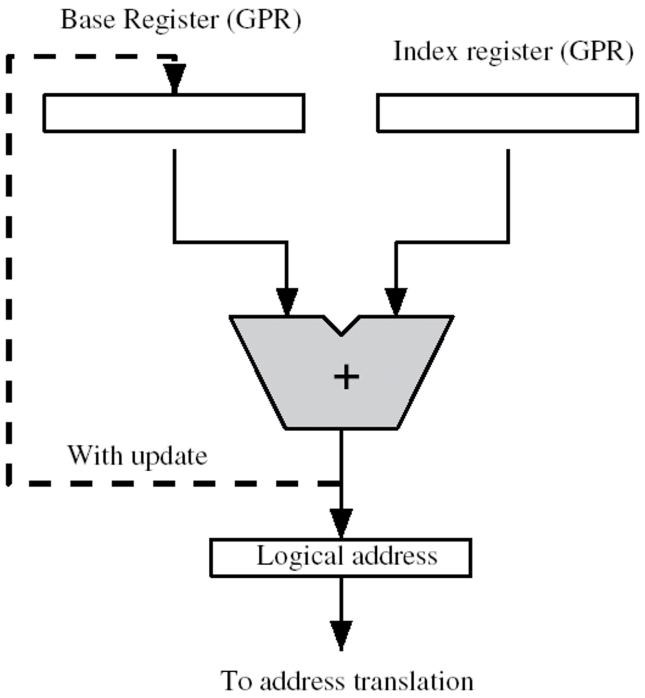
PowerPC Addressing Modes

- Load / store architecture
  - Indirect
    - Instruction includes 16 bit displacement to be added to base register (may be GP register)
    - Can replace base register content with new address
  - Indirect indexed
    - – Instruction references base register and index register (both may be GP)
    - – EA is sum of contents
- Branch address
  - Absolute
  - Relative
  - Indirect
- Arithmetic
  - Operands in registers or part of instruction
  - Floating point is register only31

PowerPC Memory Operand Addressing Modes



(a) Indirect Adressing



(b) Indirect Indexed Addressing

Instruction Formats

- Layout of bits in an instruction
- Includes opcode
- Includes (implicit or explicit) operand(s)
- Usually more than one instruction format in an instruction set

Instruction Length

- Affected by and affects:
  - Memory size
  - Memory organization
  - Bus structure
  - CPU complexity
  - CPU speed
- Trade off between powerful instruction repertoire and saving space34

Allocation of Bits

- Number of addressing modes
- Number of operands
- Register versus memory
- Number of register sets
- Address range
- Address granularity

Pentium Instruction Format