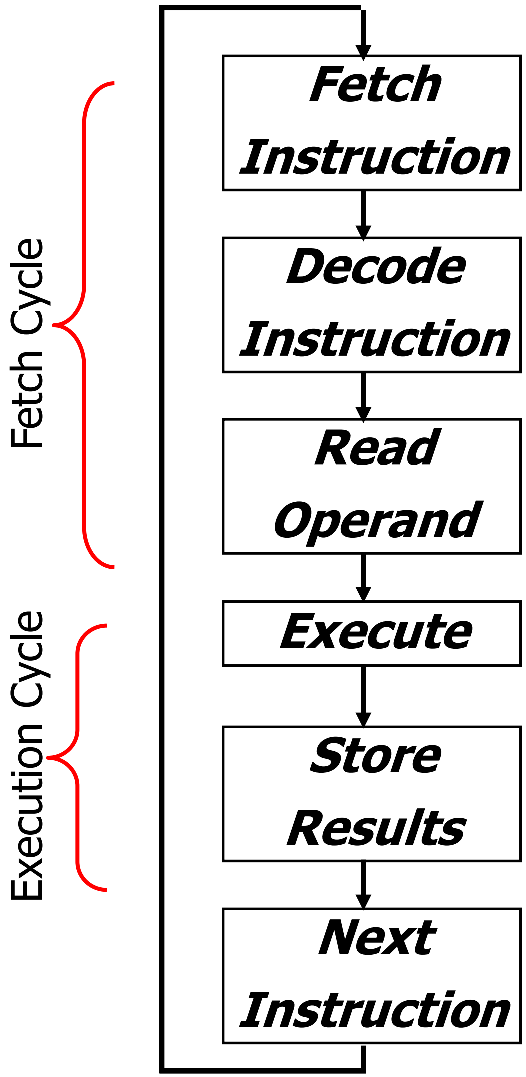
Microprocessor Systems

Dr. Gökhan İnce

Syllabus

1. Introduction, Number Systems
2. Computer Overview - Memory
3. Memory Design
4. Quiz 1, CPU overview, Instruction format
5. Addressing methods
6. Instruction types
7. Instruction types - cntd
8. Midterm Exam 1
9. Parallel communication interface
10. Serial communication interface
11. Quiz 2, Subroutines, Interrupts, Stack, Coding techniques
12. Coding examples and applications
13. Midterm Exam 2
14. Development of Microprocessor Based Designs

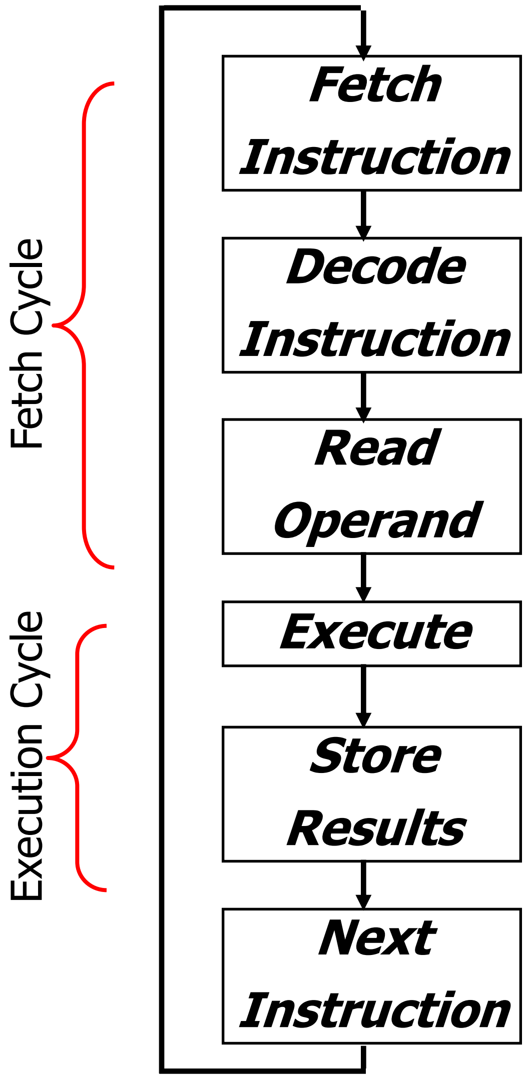
Addressing Methods



Op-code	Operand
1 or 2 bytes	0 to 3 bytes

- Addressing identifies, where to find the operand in the memory or among the registers
- During instruction decoding, the addressing mode is detected and the effective address is obtained
- There are six main addressing methods. Additional methods are derivatives
- Higher quantity and complexity of addressing modes of a CPU designates its flexibility to execute high-level programming languages

Operand Sources



- memory location
- registers
- data

Addressing Codes

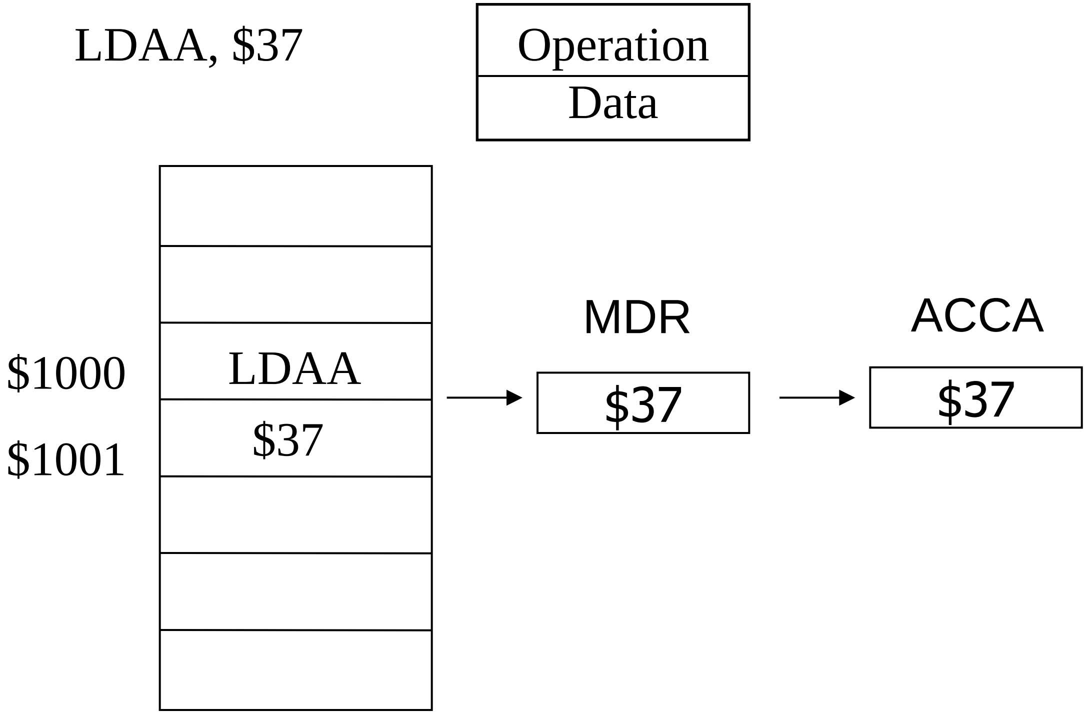
- Major Addressing Methods
  - Immediate Addressing
  - Implied Addressing (Register Addressing)
  - Direct Addressing
  - Indirect Addressing
  - Indexed Addressing
  - Relative Addressing
- Advanced Addressing Methods
  - Memory Immediate Write
  - Incremented Index Addressing
  - Decremented Index Addressing
  - Register Relative Index Addressing

Immediate Addressing

- The operand is contained in the instruction (LDA+X)
- Instruction does not specify an address location
    - Fast instructions
- Example:

  LDAB, $41

  LDS, $1000



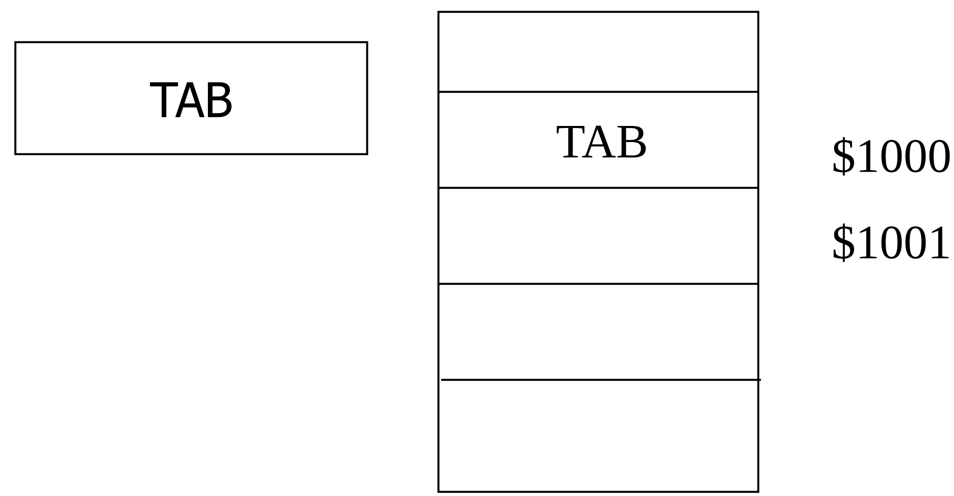
Implied (Register) Addressing

- The instruction contains the register indicator
- No addressing to the memory
- Used for register operations
- Short instruction length (ex. 1 Byte)
- Fast instructions
- Examples:

  TAB => A → B

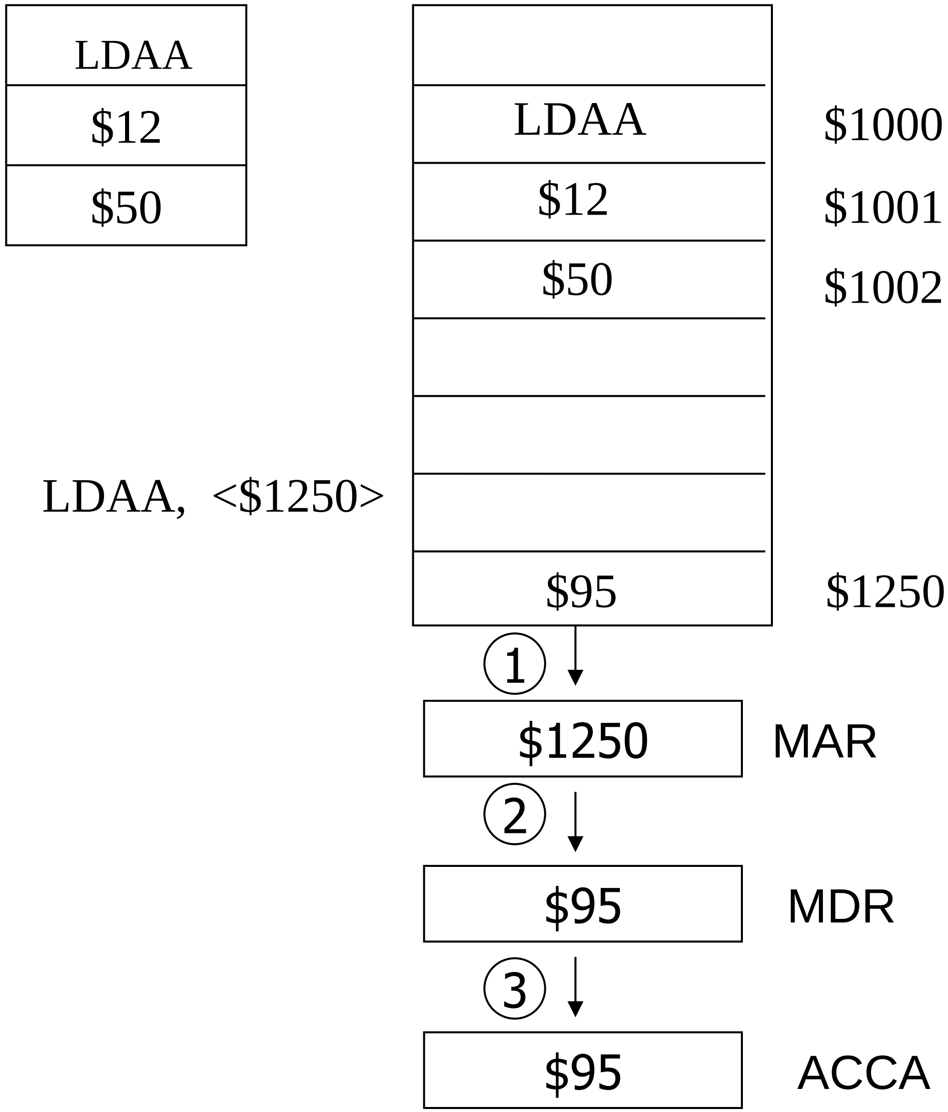
  CLRA => $00 → A

  LSRB => Logical shift right ACC B



Direct Addressing

- Instruction contains the address of the operand
- Effective address of the operand is in the instruction

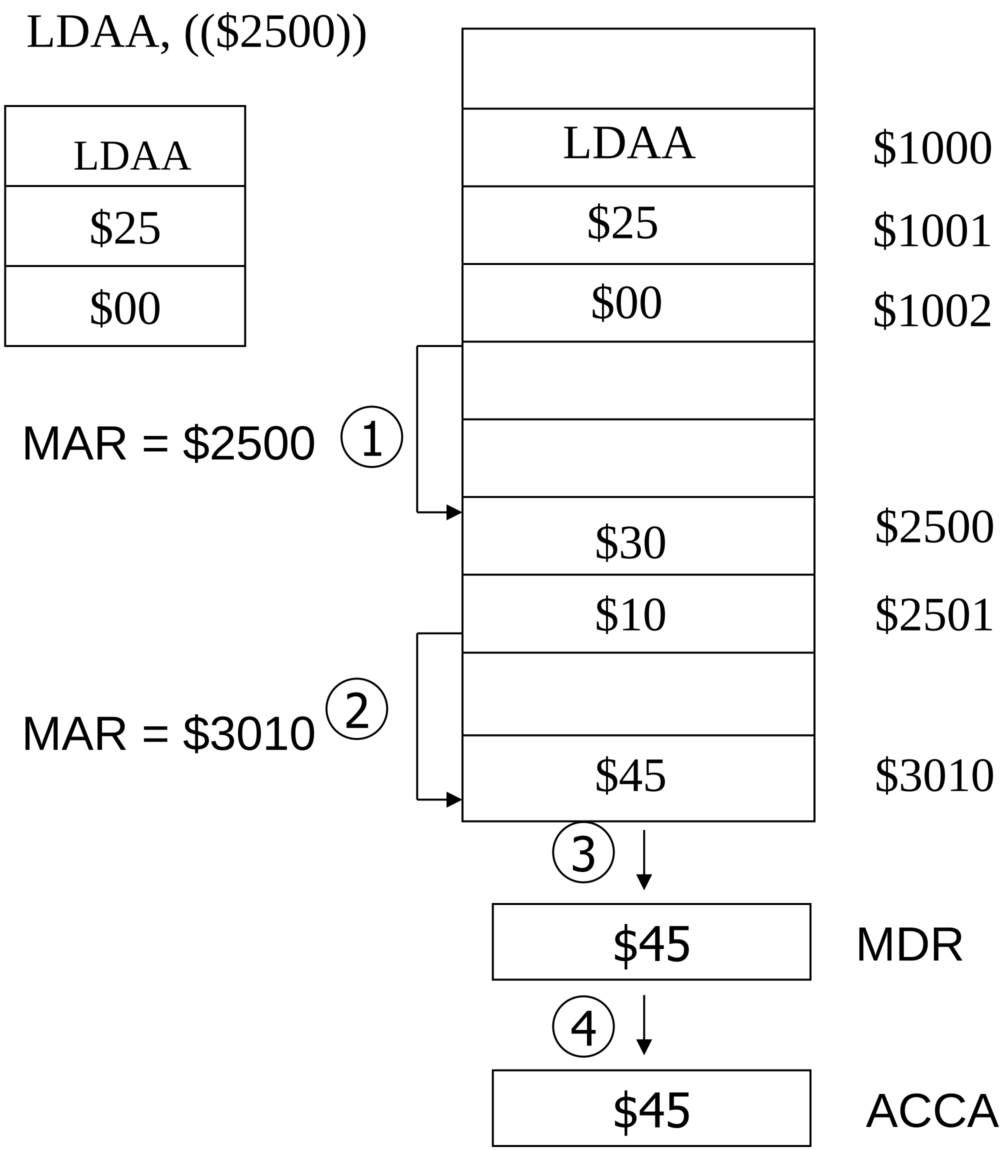


Indirect Addressing

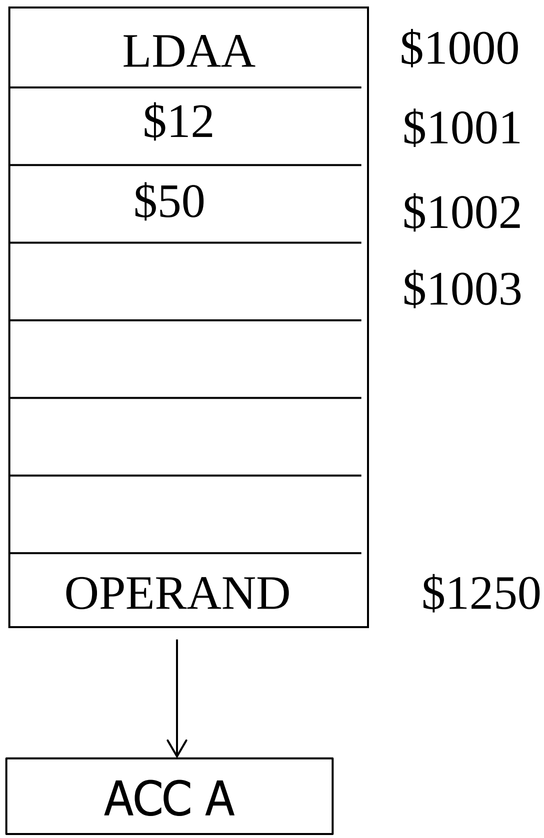
- Instruction contains the address of the operand's address
- Effective address is at the address location specified in the instruction
- Slow instructions
- Used for queues, lists, pointer type data structures
- Examples:

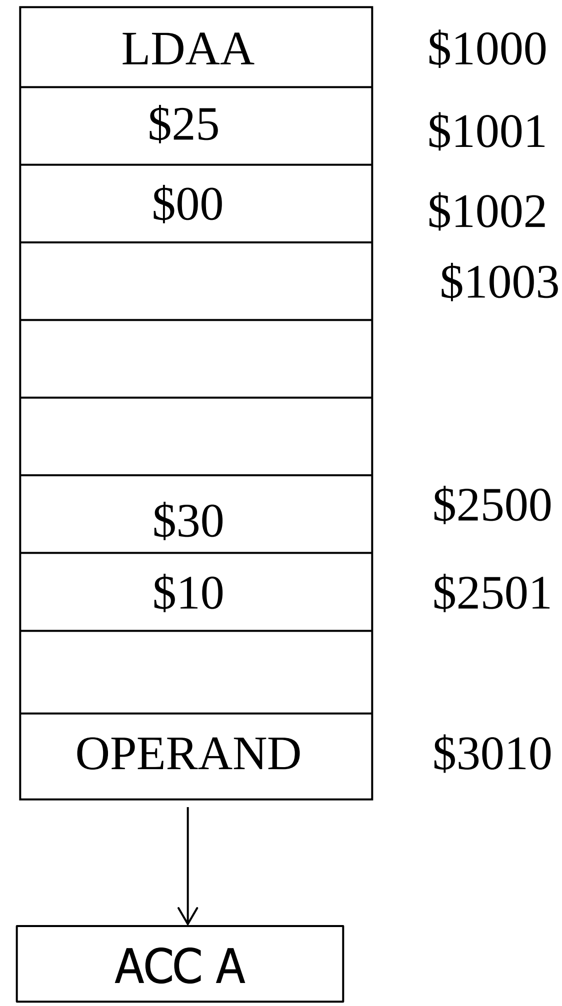
  LDAA, (($2500))

  LDAA, <CD>



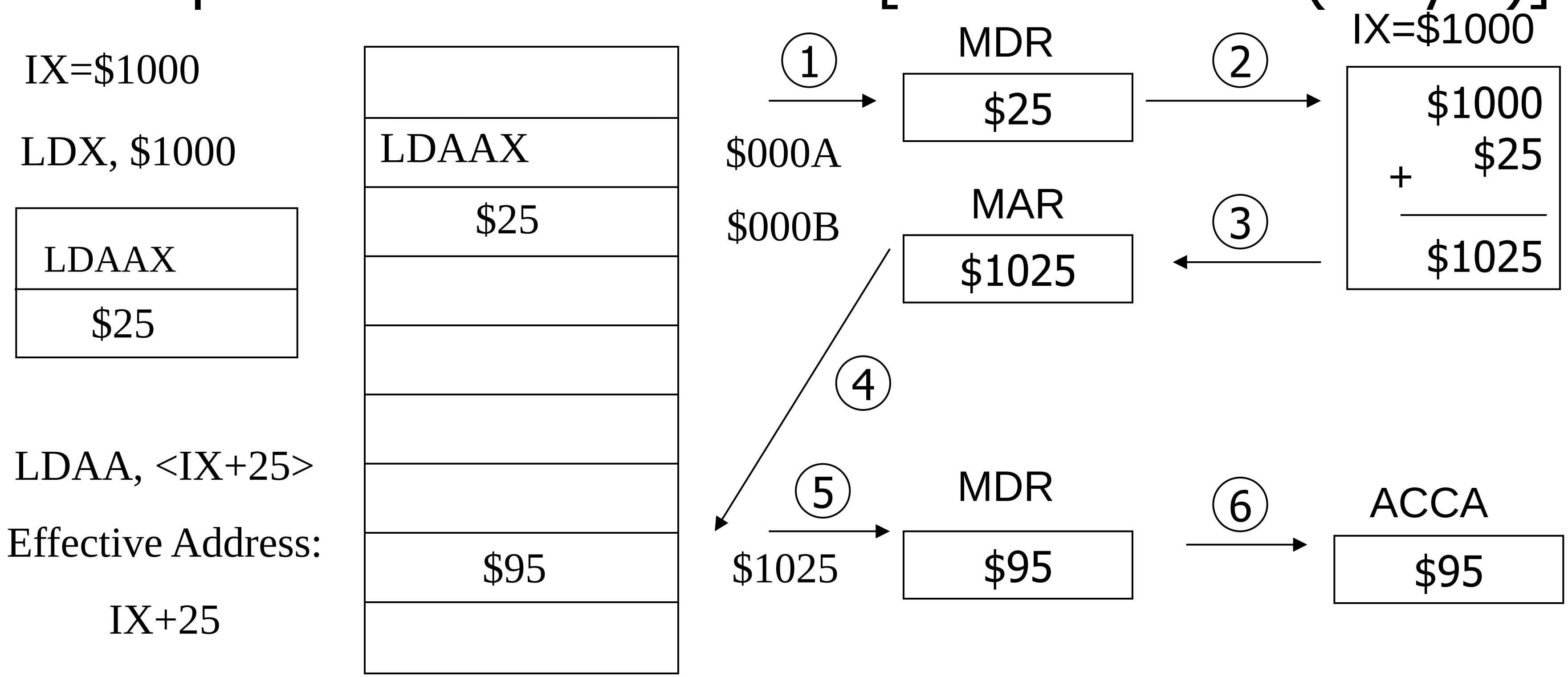
Direct – Indirect Addressing





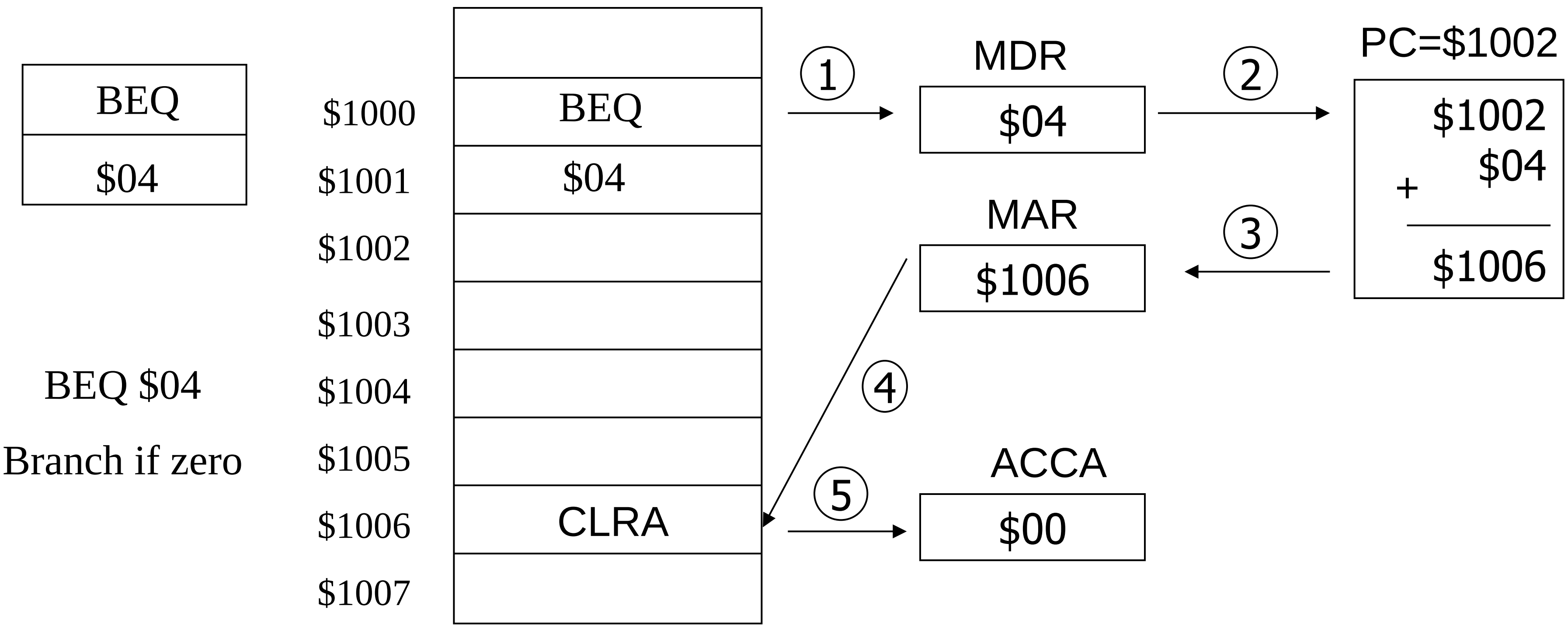
Indexed Addressing

- Index Register (IX) contains the first element of the data set.
- The operand's address = IX + [Index number (1-Byte)]



Relative Addressing

- The operand's address is relative to the
  PC = PC +/- [Step size (7 bit)]
- Used for branching instructions



Addressing Methods

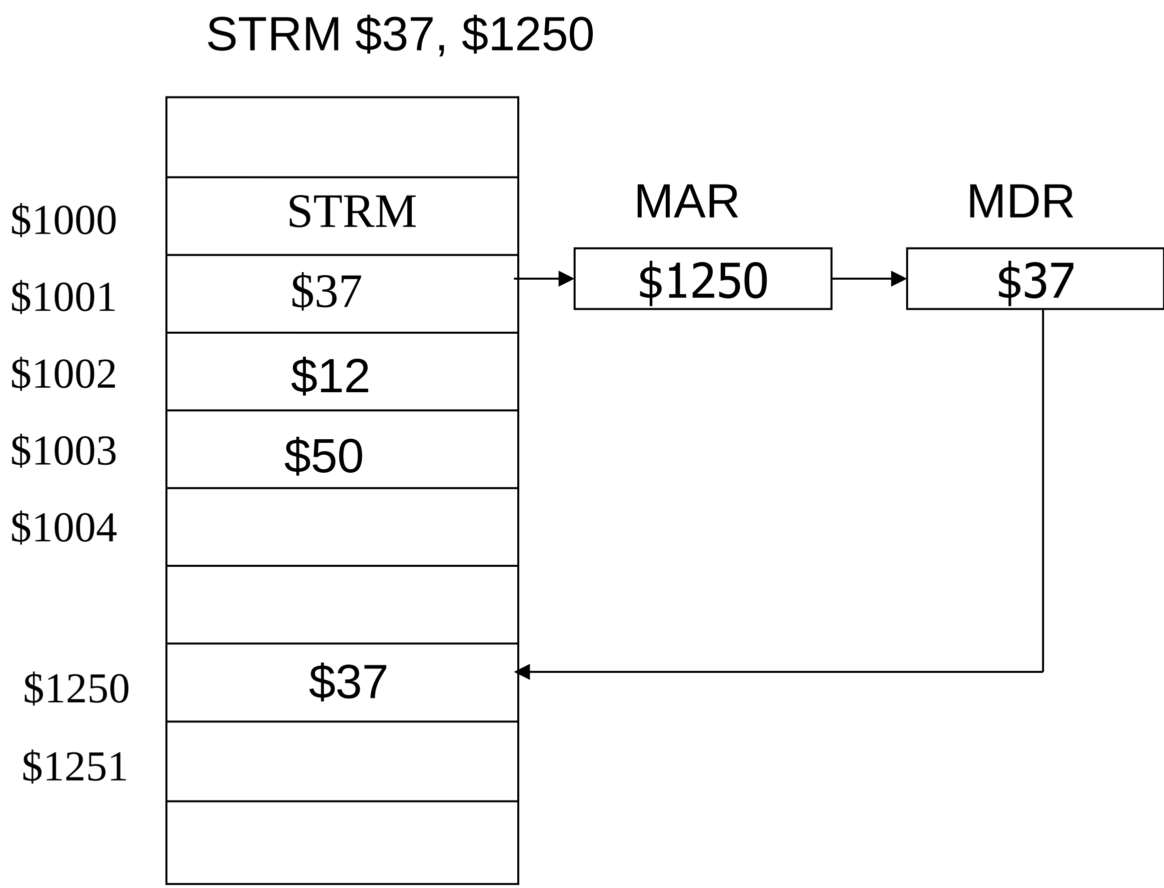
a	b	c	d	e	f	g	h	k	n	o	p	s	u	v	y	Data/address	Data/address	address

1.Octal 2.Octal 3.Octal 4.Octal 5.Octal

Addressing Method	Effective Address Formula
Immediate	Operand is in Instruction
Register	Operand is in Register
Direct Memory	E.A. = <address in instruction>
Indirect	E.A. = contents <address in instruction>
Indexed	E.A. = IX+Step

Advanced Addressing Methods

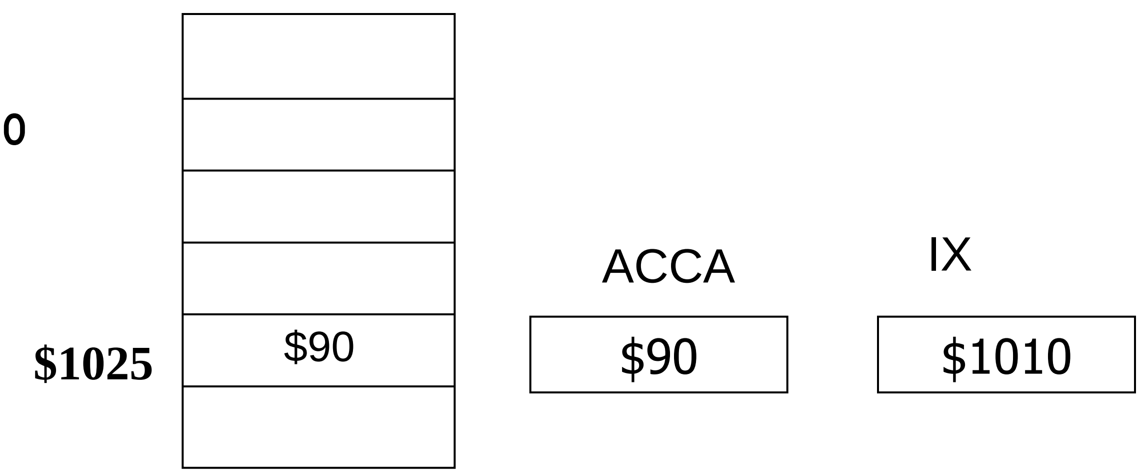
- Memory Immediate Write
  - Data written directly to a memory location
  - Accumulator, IX, SP, C, D content does not change



Advanced Addressing Methods

- Incremented Index Addressing
  - Effective address computed as IX + Index
  - Then IX is incremented by the range R [1-Byte ($00 to $FF)] provided at the instruction

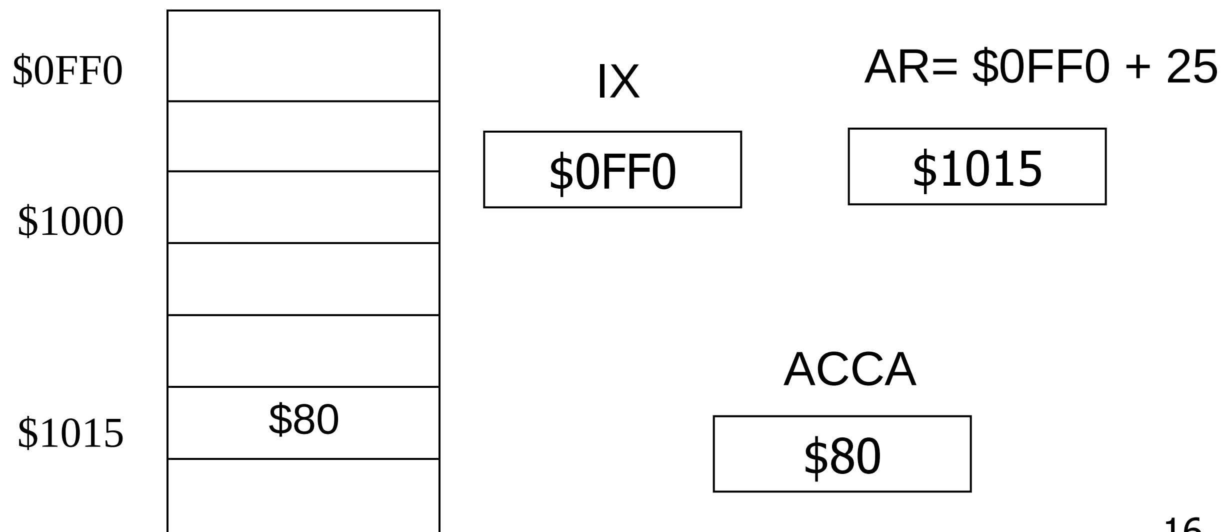
```
LDX, $1000
LDAA, <IX+25> + $10

Index = $25
Range, R = $10
```



Advanced Addressing Methods

- Decremented Index Addressing
  - First the IX is decremented by R [1-Byte number (Range $00-$FF)]
  - Then, Address of operand = IX + Index

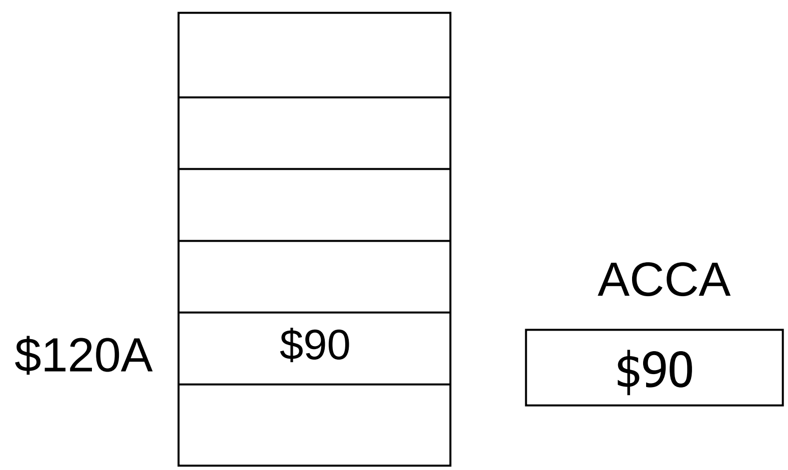
```
LDX, $1000
LDAA, <IX+25> - $10

Index = $25
Range, R = $10
```



Advanced Addressing Methods

- Register Relative Index Addressing
  - Effective address computed as the sum of General Purpose (GP) Registers [CD], Index Register [IX], and the Index provided in the instruction
  - Size of GP and IX registers should be the same

LDAA, <CD+IX+$05>
C: $02, D:$05, IX:$1000
A ⟵ <$120A>



Instruction Set Categories for Educational CPU

Data Transfer Instructions	Arithmetic-Logic-Shift Instructions		Program Control and Sequencing Instructions		Input – Output
Transfer (MOV) (AKT)	Add (ADD) (TOP)	And (AND) (VE)	Compare (CMP) (KAR)	Branch if Lower (BLO) (DEU)	Load from I/O address (LDA) (YÜK)
Load (LDA) (YÜK)	Add with carry (ADDC) (TOPE)	Or (OR) (VEYA)	Test (TST) (SIN)	Branch if Overflow (BV) (DTV)	
Store (STA) (YAZ)	Subtract (SUB) (ÇIK)	Xor (XOR) (YADA)	Unconditional Jump (JMP) (BAĞ)	Branch if Carry (BC) (DEV)	
Exchange (XCH) (XCH)	Subtract w. carry (SUBC) (ÇIKE)	Complement (COM) (TÜM)	Unconditional Branch (BR) (DAL)	Branch if Half Carry (BHC) (DYV)	Store to I/O address (STA) (YAZ)
Swap (SWA) (DĞŞ)	Multiply (MUL) (ÇAR)	Logical Shift Left (SHL) (SOL)	Conditional Jump (JMPC) (BAĞK)	Branch if Not Overflow (BNV) (DTY)	
Push (PUSH) (YIĞ)	Divide (DIV) (BÖL)	Logical Shift Right (SHR) (SAĞ)	No operation (NOP) (GEÇ)	Branch if Not Carry (BNC) (DEY)	
Pop (POP) (ÇEK)	Increment (INC) (ART)	Arithmetic Shift Right (SHRA) (SAĞİ)	Branch if Equal (BEQ) (DEE)	Branch if No Half Carry (BNHC) (DYY)	
	Decrement (DEC) (AZT)	Circular Shift Left (SHLC) (SOLD)	Branch if Not Equal (BNEQ) (DED)	Decrement, branch if not zero (ADED) (...)	
	Negative (NEG) (EKS)	Circular Shift Right (SHRC) (SAĞD)	Branch if Greater (BGT) (DEB)	Branch to Subroutine address (ALTD) (...)	
	Convert bin to BCD (DAA) (ONA)	Clear (CLR) (SİL)	Branch if Greater or Equal (BGE) (DBE)	Branch to Subroutine step (ALT) (...)	
		Set (SET) (KUR)	Branch if Less (BLT) (DEK)	Conditional Branch to Subroutine (ALTK) (...)	
			Branch if Higher (BHI) (DEİ)	Return from Subroutine (RTS) (DÖN)	
			Branch if Higher or Equal (BHS) (DİE)	Return from Interrupt (RTI) (DÖNK)	

Instruction Set Architecture (ISA) Classification

- Based on operand location
- Type and size of operands
- Memory addressing
- Addressing modes
- **Operations in the instruction set**
- Instruction set encoding

Instruction Types

- Arithmetic
  - Add, Subtract, Multiply, Divide
- Logical
  - Shift, Rotate
- Data Movement
  - Load, Store
- Flow Control
  - Unconditional Branch
  - Conditional Branch

Instruction Types

- Data Transfer Instructions
  - Operations that move data from one place to another
  - These instructions don’t actually modify the data, they just copy it to the destination
- Data Operation Instructions
  - Unlike the data transfer instructions, the data operation instructions do modify the values of data
  - They typically perform some operation using one or two data values (operands) and store the result
- Program Control Instructions
  - Jump or branch instructions are used to go to another part of the program; the jumps can be absolute (always taken) or conditional (taken only if some condition is met)
  - Specific instructions that can generate interrupts (software interrupts)

Transfer Instructions

- Data Transfer: moves the contents of one register into another.

MOV R_i, R_j $R_i \leftarrow R_j$

MOV A,B

MOV A,C

MOV B,CCR

MOV IX,SP

Sometimes T is used: e.g., TXS ->T IX, SP

Transfer Instructions

- Load: The load instruction copies the contents of a memory location or places an immediate value into an accumulator or a register. Memory contents are not changed.

LDA R_i, <ADDRESS>	$R_i \leftarrow$ <ADDRESS>
LDA A, <$1000>	ACCA $\leftarrow$ <$1000>
LDA C, <$2000>	C $\leftarrow$ <$2000>
LDA B, $25	B $\leftarrow$ $25
LDA IX, <$A000>	IX $\leftarrow$ <$A000> + <$A001>
LDA IX, $C000	IX $\leftarrow$ $C000

Transfer Instructions

- Store: Store instructions copy the contents of a CPU register into a memory location. The contents of the accumulator or CPU register are not changed.

STA R_i, <MEMORY>	<MEMORY> $\leftarrow$ R_i
STA A, \$1000	\$1000 $\leftarrow$ ACC A
STA C, \$E000	\$E000 $\leftarrow$ C
STA IX, \$C000	\$C000 + \$C001 $\leftarrow$ IX

Transfer Instructions

- Exchange: Exchange instructions exchange the contents of pairs of registers or accumulators.

XCH R_i, R_j	$R_i \longleftrightarrow R_j$
XCH A, B	ACC A $\longleftrightarrow$ ACC B
XCH C, D	C $\longleftrightarrow$ D
XCH IX, SP	IX $\longleftrightarrow$ SP

- Swap: The bits are divided into groups of four. Swap insructions are used to swap the contents of the first group with the second one.

SWA R_i,	$R_i[D_3,D_2,D_1,D_0 - D_7,D_6,D_5,D_4]$
SWA A	

Transfer Instructions

- Push: The contents of the accumulators are pushed onto the stack

PUSH A	push the contents of A onto stack
PUSH B	push the contents of A onto stack

- Pop: Pop instructions are used to read data into the specified accumulator from the stack

POP A	data on top of the stack is retrieved to ACCA
POP B	data on top of the stack is retrieved to ACCA

Arithmetic Instructions

- Add: Add instructions are used to add the contents specified, either in an accumulator or in a register or in memory location, into the accumulator.The result is stored in the accumulator.

ADD A, B	ACCA $\leftarrow$	ACCA+ACCB
ADD A, R_i	ACCA $\leftarrow$	ACCA+R_i
ADD A, DATA	ACCA $\leftarrow$	ACCA+DATA
ADD A, <ADDRESS>	ACCA $\leftarrow$	ACCA+ <ADDRESS>

- Add with Carry: Add operation is performed by including the caary flag.

ADDC A, B	ACCA $\leftarrow$	ACCA+ACCB+C
ADDC A, R_i	ACCA $\leftarrow$	ACCA+R_i+C
ADDC A, DATA	ACCA $\leftarrow$	ACCA+DATA+C
ADDC A, <ADDRESS>	ACCA $\leftarrow$	ACCA+<ADDRESS>+C

Add Instruction

ADD A, B	ACC A ← ACC A + ACC B
ADDC A, B	ACC A ← ACC A + ACC B + C
ADD A, $25	ACC A ← ACC A +$25
ADD A, <$1000>	ACC A ← ACC A + <$1000>
ADDC A, <IX+10>	ACC A ← ACC A + <IX+10> + C

Add Instruction

- This program reads two numbers (N1,N2) from memory, adds them and stores the result into memory.

  N1 + N2 ->S

  N1->$c200

  N2->$c201

  S->$c202
- The program starts at memory location $C000.

	Address	Contents
LDA A, <$C200>	C000	00
	C001	20
	C002	C2
	C003	00
LDA B, <$C201>	C004	00
	C005	21
	C006	C2
	C007	01
ADD A,B	C008	43
	C009	01
STA A, <$C202>	C00A	01
	C00B	20
	C00C	C2
	C00D	02
	:	:
N1	C200	28
N2	C201	55
S	C202	7D

Add Instruction

- Example: Add two numbers X=$40A0 ve Y=$1BC1

<10>	<11>	X	<10>:0100 0000
<12>	<13>	Y	<11>:1010 0000
+			<12>:0001 1011
<14>	<15>	RESULT	<13>:1100 0001

LDA A, <$0010>	ACCA	←	0100 0000
LDA B, <$0011>	ACCB	←	1010 0000
ADD B, <$0013>	ACCB	←	0110 0001 C=1(carry)
ADDC A, <$0012>	ACCA	←	0101 1100
STA A, <$0014>	$0014	←	0101 1100
STA B, <$0015>	$0015	←	0110 0001

Subtract Instruction

- Subtract: Substract instructions are used to subtract the specified contents, in an acuumulator or in a register or in the memory location, from the contents of the accumulator. The result is stored in the accumulator.

SUB A, B	ACCA $\longleftarrow$	ACCA-ACCB
SUB A, R_i	ACCA $\longleftarrow$	ACCA-R_i
SUB A, DATA	ACCA $\longleftarrow$	ACCA-DATA
SUB A, <ADDRESS>	ACCA $\longleftarrow$	ACCA- <ADDRESS>

- Subtract with Borrow

SBC A, B	ACCA $\longleftarrow$	ACCA-ACCB-C
SBC A, R_i	ACCA $\longleftarrow$	ACCA-R_i-C
SBC A, DATA	ACCA $\longleftarrow$	ACCA-DATA-C
SBC A, <ADDRESS>	ACCA $\longleftarrow$	ACCA-<ADDRESS>-C

Arithmetic Instructions

- Multiplication:
  - The multiplicand is in ACCA.
  - The multiplier is in ACCB or in an 8 bit register or in a memory location or an immediate data.
  - The result is stored in ACCA and ACCB.
  - The multiplicand and multiplier are unsigned.

MUL A, B	ACCA+ACCB	$\leftarrow$	ACCA * ACCB
MUL A, R_i	ACCA+ACCB	$\leftarrow$	ACCA * R_i
MUL A, DATA	ACCA+ACCB	$\leftarrow$	ACCA * DATA
MUL A, <ADDRESS>	ACCA+ACCB	$\leftarrow$	ACCA * <ADDRESS>

Arithmetic Instructions

- Division:
  - The dividend is in ACCA and ACCB pair.
  - The divisor is an 8 bit register or a memory location or an immediate data.
  - The quotient is in ACCA and ACCB pair and the remainder is in C.
  - The dividend and the divisor are unsigned numbers.
  - If the divisor is 0, the overflow flag in CCR is set.

DIV AB, R_i	ACCA + ACCB	$\leftarrow$	<ACCA + ACCB> / R_i
DIV AB, DATA	ACCA + ACCB	$\leftarrow$	<ACCA + ACCB> / DATA
DIV AB, <ADDRESS>	ACCA + ACCB	$\leftarrow$	<ACCA +ACCB>/<ADDRESS>

Example

- The 16 bit number in memory locations $1000 and $1001 will be divided to the number in memory location $1002. The result and the remainder will be stored in memory locations $1005-$1006 and $1007 respectively.

```
START   LDA IX, $1000
        LDA A, <IX+0> + $01
        LDA B, <IX+0> + $01
        DIV AB, <IX+0>
        STA AB, <$1005>
        STA C, <$1007>
```

Example

- The operands in memory locations $1000 and $1001 will be multiplied, then divided to the operand in memory location $1002. The result will be stored at memory locations $1005-$1006, remainder will be stored at memory location $1007

```
START   LDA IX, $1000              IX<-$1000
        LDA A, <IX+0> + $01        IX<-$1001
        MUL A, <IX+0> + $01        IX<-$1002
        DIV AB, <IX+0>             IX<-$1002
        LDA IX, $1005              IX<-$1005
        STA AB, <IX+0> + $02       IX<-$1007
        STA C, <IX+0>
```