Principles of Computers
13th Lecture

http://d3s.mff.cuni.cz/~jezek

Pavel Ježek, Ph.D.
pavel.jezek@d3s.mff.cuni.cz

CHARLES UNIVERSITY IN PRAGUE
faculty of mathematics and physics
procedure P1(a : word;
    b : longword);

$0A50: ... ← IP

... ...
ret

begin ...
    push 00000007h
    push 0005h
$0900: call 00000A50h
$0905: SP := SP + (4+2)
        nop ...
end.
procedure P1(a : word;
    b : longword);
$0A50: ... ← IP
... ...
ret
begin
... push 00000007h
push 0005h
$0900: call 00000A50h
$0905: SP := SP + (4+2)
nop...
end.
```pascal
var
  x : word;

procedure P1(a : word;
             b : longword);

begin
  ...$0A50: ...
  ...
  ...
  ret
  ...
  x := a;
  ...
  ...push 00000007h
  push 005h
$0900: call 00000A50h
$0905: SP := SP + (4+2)
  nop
  ...
end.
```
This code is still just **pseudocode** with Pascal-like syntax! The “\(? := (^{\text{word}}(? + ?))^{\wedge}\)” command is in general **not a valid instruction**. It is also **not a valid Pascal program**, because we **cannot** directly access SP register in Pascal code (or any other CPU register).

The whole \(x := (^{\text{word}}(\text{SP} + 4))^{\wedge}\) expression has to be further compiled into an actual instruction sequence yet – see lectures later in this semester.
var
  x : word;

procedure P1(a : word;
            b : longword);

$x0A50$: ...
  (^word($0000B00)) := (^word(SP + 4))

begin ...
  x := a;
  push 00000007h
  push 0005h
  $0900$: call 00000A50h
  $0905$: SP := SP + (4+2)
  nop ...
end.
var
  x : word;

procedure P1(a : word;
  b : longword);

$0A50: \ldots \rightarrow IP

\text{begin}
  \ldots
  \text{push 00000007h}
  \text{push 0005h}
  \text{\$0900: call 00000A50h}
  \text{\$0905: SP := SP + (4+2)}
  \text{nop}
  \ldots
\text{end.}

b := b + (a + 42);
procedure P1(a : word;
    b : longword);
var
    loc1, loc2 : word;
$0A50$: ... ← IP
    ...
    ...
    ret
begin
    ...
    push 00000007h
    push 0005h
$0900$: call 00000A50h
$0905$: SP := SP + (4+2)
    nop
    ...
end.
procedure P1(a : word;
    b : longword);
var
    loc1, loc2 : word;
$0A50$: SP := SP – (2+2) ← IP
    ...
    ...
    ret
begin
    ...
    push 00000007h
    push 0005h
$0900$: call 00000A50h
$0905$: SP := SP + (4+2)
    nop
    ...
end.
procedure P1(a : word;
             b : longword);
var
  loc1, loc2 : word;
$0A50: SP := SP \(-\)(2+2)
...
...
ret \leftarrow IP
begin
...
push 00000007h
push 0005h
$0900: call 00000A50h
$0905: SP := SP + (4+2)
nop
...
end.
begin
push $00000007h
push $0005h
$0A50: SP := SP – (2+2)
$0900: call $00000A50h
$0A50: SP := SP + (2+2) ← IP
ret
...
$0000101A
00 00 00
$00001016
00 00 00
$00001014
00 00 00 09 05
$00001010
xx
$0000100E
xx
$00001012
xx
$0000100C
xx
...

procedure P1(a : word; b : longword);
var
loc1, loc2 : word;
end.

procedure argument
b
procedure argument
a
ger

local variable loc1
local variable loc2
return address from procedure P1
to main program
procedure argument
procedure P1(a : word; b : longword);
    var
        loc1, loc2 : word;
    begin
        push 00000007h
        push 0005h
        call 00000A50h
        SP := SP + (4+2)
        nop
        ...
procedure P1(a : word;  
    b : longword); 
var 
    loc1, loc2 : word; 
$0A50: SP := SP – (2+2)$  
    ...  
    SP := SP + (2+2)  
ret  
begin  
    ...  
push 00000007h  
push 0005h  
$0900: call 0000A50h$  
$0905: SP := SP + (4+2)$  
nop  
    ...  
end.
function F1(a : word;  
    b : longword  
  ) : word;

var  
  loc1, loc2 : word;

$0A50: SP := SP - (2+2)$

SP := SP + (2+2)

ret

var  
  x : word;

begin ...

push 00000007h
push 0005h

$0900: call 0000A50h$

$0905: ($00000B00)^ := ?$

SP := SP + (4+2)

nop ...

end.

x := F1(5, 7);
function F1(a : word;
   b : longword
 ) : word;
var
   loc1, loc2 : word;

$0A50: SP := SP – (2+2)
...
SP := SP + (2+2)
ret

var
   x : word;
begin
...
push 00000007h
push 0005h
SP := SP - 2
$0900: call 00000A50h
$0905: ($00000B00)^ := ?
SP := SP + (4+2+2)
nop
...
end.

x := F1(5, 7);
function F1(a : word;
   b : longword
  ) : word;
var
  loc1, loc2 : word;
$0A50: SP := SP – (2+2)
... (SP + 8)^ := retval
SP := SP + (2+2)
ret
var
  x : word;
begin
  ...
push 00000007h
push 0005h
SP := SP - 2
$0900: call 00000A50h
$0905: ($00000B00)^ := ?
SP := SP + (4+2+2)
nop
...
x := F1(5, 7);
end.
function F1(a : word;
    b : longword
  ) : word;
var
  loc1, loc2 : word;
$0A50: SP := SP – (2+2)
... (SP + 8)^ := retval
  SP := SP + (2+2)
  ret
var
  x : word;
begin
  ...
push 00000007h
push 0005h
  SP := SP - 2
$0900: call 00000A50h
$0905: (^word($00000B00))^ := SP^  <- IP
SP := SP + (4+2+2)
nop
...
end.

x := F1(5, 7);
```
program PascalProgram;

type
  PProc = procedure;

procedure P1;
begin
  \[ \alpha \]
  jmp back \equiv \text{ret}
end;

procedure P2;
begin
  \[ \beta \]
  jmp back \equiv \text{ret}
end;

var
  i : word;
  ptr : PProc;
  j : word;

begin
  \[ \gamma_1 \]
  ptr := @P1;

  \[ \gamma_2 \]
  ptr;
  P2;

  \[ \gamma_3 \]
  end.
end.
```
program PascalProgram;

type
  PProc = procedure;

procedure P1;
begin
  α
end;  jmp back ≡ ret

procedure P2;
begin
  β
end;  jmp back ≡ ret

var
  i : word;
  ptr : PProc;
  j : word;

begin
  γ
  ptr := @P1;
  ptr;  P2;
  γ
  end
end.

CALL 15
indir FF
ptr addr?

$00001300
$0000136
$00001306
$00001000
$00001000
...
...
program PascalProgram;

type
    PProc = procedure;

procedure P1;
begin
    \( \alpha \) - A
    jmp back \( \equiv \) ret
end;

procedure P2;
begin
    \( \beta \) - B
    jmp back \( \equiv \) ret
end;

var
    i : word;
    ptr : PProc;
    j : word;

begin
    \( \nu_1 \) - C1
    ptr := @P1;

    ptr;

    P2;

    \( \nu_2 \) - C2
end.
program PascalProgram;

type
  PProc = procedure;

procedure P1;
begin
  \alpha
end;

procedure P2;
begin
  \beta
end;

var
  i : word;
  ptr : PProc;
  j : word;

begin
  \gamma
  ptr := @P1;
  ptr;
  P2;
  \gamma
end.

\begin{verbatim}
main program
\end{verbatim}
program PascalProgram;

type
  PProc = procedure;

procedure P1;
begin
  jmp back = ret
end;

procedure P2;
begin
  jmp back = ret
end;

var
  i : word;
  ptr : PProc;
  j : word;

begin
  ptr := @P1;
  CALL E8 $00001306
  jmp indir $00001300
end.

main program

procedure P1

procedure P2

call A

$00007A00

$00002100

$00002000

$00001300

$00001306

$00001300

$00001000

$00000000

$00000000
program PascalProgram;

type
  PProc = procedure;

procedure P1;
begin
  \( \alpha \)
  \text{jmp back \equiv ret}
end;

procedure P2;
begin
  \( \beta \)
  \text{jmp back \equiv ret}
end;

var
  i : word;
  ptr : PProc;
  j : word;

begin
  \( V_1 \)
  ptr := @P1;
  ptr;
  P2;
  \( V_2 \)
end.

\text{main program}

\begin{align*}
\text{CALL E8} & \quad $00001306 \\
\text{CALL 15} & \quad $00001300 \\
\text{indir FF} & \quad $00001300 \\
\text{ptr addr?} & \\
\text{ptr addr?} & \\
\end{align*}
program PascalProgram;

type
  PProc = procedure;

procedure P1;
begin
  \[\alpha\]  \(\text{jmp } \text{back} \equiv \text{ret}\)
end;

procedure P2;
begin
  \[\beta\]  \(\text{jmp } \text{back} \equiv \text{ret}\)
end;

var
  i : word;
  ptr : PProc;
  j : word;

begin
  \[\gamma\]
  \[\gamma_1\]  \(\text{ptr} := \&P1;\)
end.

main program

\[\nu_1\]  \(\text{ptr; \ P2; \ }\nu_2\)
How a Debugger Works?
push dword 0
push dword ptr [00B9105Bh]
push dword ptr [00B9105Bh]
push dword 40h
code and data of DEBUGGER

code and data of debugged application (= debugee)

push dword 0
push dword ptr [00B9105Bh]
push dword ptr [00B9105Bh]
push dword 40h
code and data of DEBUGGER

code and data of debugged application (= debugee)

push dword 0

push dword ptr [00B9105Bh]

push dword 0

push dword ptr [00B9105Bh]

push dword 40h
code and data of DEBUGGER

00
6A
B9
10
1A
68
$00B91007

push dword ptr [00B9105Bh]

code and data of debugged application (= debugee)

00
6A
B9
10
5B
68
40
6A
$00B91000

push dword 0

$00B9100C

push dword ptr [00B9105Bh]

$01000000

push dword 40h

$00B91002

push dword ptr [00B9105Bh]

$00B91000
code and data of DEBUGGER

... $01000000
...
00
6A
$00B9100C
00
B9
10
1A
68
$00B91007
00
B9
10
5B
68
$00B91002
40
6A
$00B91000
...

code and data of debugged application (= deuggee)

push dword 0
push dword ptr [00B9105Bh]
push dword ptr [00B9105Bh]
push dword 40h
code and data of debugged application (= debugee)

- push dword $00B91002
- push dword 40h
- push dword $00B91000

code and data of DEBUGGER

- push dword $00B91007
- push dword ptr [00B9105Bh]

- push dword $00B9100C
- push dword ptr [00B9105Bh]

- push dword 0

variable holding copy of original app’s code

- push dword 40h

CALL to entrypoint

code and data of debugged application (= debugee)

variable address of debugger step function

variable holding copy of original app’s code

CALL to entrypoint

code and data of DEBUGGER

push dword 0

push dword ptr [00B9105Bh]

push dword ptr [00B9105Bh]

call [0100FFF0h]

push dword ptr [00B9105Bh]

push dword 40h

push dword 0

push dword ptr [00B9105Bh]

push dword ptr [00B9105Bh]
CALL to entrypoint

CALL to entrypoint

push dword 0

push dword ptr [00B9105Bh]

push dword ptr [00B9105Bh]

call [0100FFF0h]

code and data of debugged application (= debugee)

code and data of DEBUGGER

variable holding copy of original app’s code

variable address of debugger step function

SP->

retaddr
from
main
program

EIP
CALL to entrypoint

program

\[ \text{code and data of debugee} \]

\[ \text{variable holding copy of original app's code} \]

\[ \text{variable address of debugger step function} \]

\[ \text{CALL to entrypoint} \]
Execute main debugger loop (Update/Draw cycle to display UI)

- variable holding copy of original app's code
- variable address of debugger step function

Save state of application (e.g. push all registers to stack)

1. push dword 40h
2. push dword [00B9105Bh]
3. call [0100FFF0h]
4. push dword ptr [00B9105Bh]
5. push dword ptr [00B9105Bh]

Code and data of debugged application (= debugee)

- code and data of DEBUGGER
- variable holding copy of original app's code
- variable address of debugger step function
Execute main debugger loop (Update/Draw cycle to display UI)

variable holding copy of original app’s code

variable address of debugger step function

save state of application (e.g. push all registers to stack)

CALL to entrypoint

push dword 0

push dword ptr [00B9105Bh]

call [0100FFF0h]

push dword ptr [00B9105Bh]

push dword ptr [00B9105Bh]

push dword 40h

push dword 0

push dword ptr [00B9105Bh]
Execute main debugger loop (Update/Draw cycle to display UI)

save state of application (e.g. push all registers to stack)

CALL to entrypoint

variable holding copy of original app's code

variable address of debugger step function

code and data of DEBUGGER

code and data of debugged application (= deuggee)

retaddr from main program

SP->
dword

retaddr
saved
registers

push dword 0

push dword ptr [00B9105Bh]

push dword ptr [00B9105Bh]

push dword 40h

push dword 0

push dword 00B9100C

...
CALL to entrypoint

Execute main debugger loop (Update/Draw cycle to display UI)

variable holding copy of original app’s code

variable address of debugger step function

Save state of application (e.g. push all registers to stack)

push dword 0

push dword ptr [00B9105Bh]

push dword ptr [00B9105Bh]

push dword 40h

push dword 00

push dword 40

push dword 00

push dword [00B9105Bh]

push dword 00

push dword 0B

push dword 08

push dword 00

push dword 00

push dword 00

push dword 00

push dword 6A

push dword 00

push dword 10

push dword 1A

push dword 68

push dword 10

push dword 5B

push dword 68

push dword 40

push dword 6A
Execute main debugger loop (Update/Draw cycle to display UI)

save state of application (e.g. push all registers to stack)

push dword 0

call [0100FFF0h]

push dword ptr [00B9105Bh]

push dword ptr [00B9105Bh]

push dword 40h

push dword 0

push dword ptr [00B9105Bh]
Execute main debugger loop (Update/Draw cycle to display UI)

Save state of application (e.g., push all registers to stack)

Push DWORD 40h

Push DWORD PTR [00B9105Bh]

Push DWORD 0

Call [0100FFF0h]

Push DWORD PTR [00B9105Bh]

Push DWORD PTR [00B9105Bh]
CALL to entrypoint

execute main debugger loop (Update/Draw cycle to display UI)

variable holding copy of original app's code
variable address of debugger step function
restores state of application & patch return address on stack by -6 & jump back

execute main debugger loop (Update/Draw cycle to display UI)
save state of application (e.g. push all registers to stack)
call [0100FFF0h]
push dword ptr [00B9105Bh]
push dword ptr [00B9105Bh]
push dword 0

push dword ptr [00B9105Bh]
push dword 40h
call [0100FFF0h]
push dword ptr [00B9105Bh]
push dword ptr [00B9105Bh]
push dword 0

save state of application (e.g. push all registers to stack)
call [0100FFF0h]
push dword ptr [00B9105Bh]
patch return address on stack by -6 & jump back

Execute main debugger loop (Update/Draw cycle to display UI)

variable holding copy of original app's code
variable address of debugger step function
restores state of application & patch return address on stack by -6 & jump back

execute main debugger loop (Update/Draw cycle to display UI)
save state of application (e.g. push all registers to stack)
call [0100FFF0h]
push dword ptr [00B9105Bh]
push dword ptr [00B9105Bh]
push dword 0

save state of application (e.g. push all registers to stack)
call [0100FFF0h]
push dword ptr [00B9105Bh]
patch return address on stack by -6 & jump back

Execute main debugger loop (Update/Draw cycle to display UI)

variable holding copy of original app's code
variable address of debugger step function
restores state of application & patch return address on stack by -6 & jump back
Execute main debugger loop (Update/Draw cycle to display UI)

save state of application (e.g. push all registers to stack)

CALL to entrypoint

code and data of debugged application (= debugee)

variable holding copy of original app’s code

variable address of debugger step function

push dword ptr [00B9105Bh]

push dword 0

call [0100FFF0h]

push dword ptr [00B9105Bh]

push dword 40h

push dword 0
code and data of \textbf{DEBUGGER} \hfill code and data of debugged application (= debugee)

\begin{itemize}
\item \textbf{variable holding copy of original app's code}
\item variable address of debugger step function
\item \textbf{Execute main debugger loop (Update/Draw cycle to display UI)}
\item \textbf{save state of application (e.g. push all registers to stack)}
\end{itemize}

\texttt{CALL to entrypoint}
Execute main debugger loop (Update/Draw cycle to display UI)

Save state of application (e.g. push all registers to stack)

Push dword ptr [00B9105Bh]

Push dword 0

Call [0100FFF0h]

Push dword ptr [00B9105Bh]

Push dword ptr [00B9105Bh]

Push dword 40h

Push dword 0

Push dword 0

Push dword 15FF

Variable holding copy of original app’s code

Variable address of debugger step function

Restore state of application & jump back

Code and data of debugged application (= debugee)

Code and data of DEBUGGER

CALL to entrypoint

EIP
Execute main debugger loop (Update/Draw cycle to display UI)

variable holding copy of original app’s code

variable address of debugger step function

RESTORE state of application & jump back

save state of application (e.g. push all registers to stack)

code and data of debugged application (= debugee)

code and data of DEBUGGER

CALL to entrypoint

push dword 0

call [0100FFF0h]

push dword ptr [00B9105Bh]

push dword ptr [00B9105Bh]

push dword 40h

push dword 0

Execute main debugger loop (Update/Draw cycle to display UI)

- variable holding copy of original app’s code
- variable address of debugger step function

Save state of application (e.g. push all registers to stack)

- push dword 0
- call [0100FFF0h]

- push dword ptr [00B9105Bh]
- push dword ptr [00B9105Bh]
- push dword ptr [00B9105Bh]

- push dword 40h
- push dword 0

- push dword ptr [00B9105Bh]
Stepping in Higher Level Programming Language (e.g. Pascal)

<table>
<thead>
<tr>
<th>A.pas</th>
<th>A’s machine code</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>I1</td>
</tr>
<tr>
<td>C2</td>
<td>I2</td>
</tr>
<tr>
<td>C3</td>
<td>I3</td>
</tr>
</tbody>
</table>

A’s machine code:
- I1
- I2
- I3
- I4
- I5
- I6
Stepping in Higher Level Programming Language (e.g. Pascal)

- A.pas
- C1
- C2
- C3

A’s machine code
- I1
- I2
- I3
- I4
- I5
- I6
Stepping in Higher Level Programming Language (e.g. Pascal)
Typical ISA Arithmetic Instructions

MIPS: $a := b \ op \ c$

x86, 6502: $a := a \ op \ b$
6502 Registers (Accumulator Architecture)

<table>
<thead>
<tr>
<th>Register</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>70</td>
</tr>
<tr>
<td>X</td>
<td>70</td>
</tr>
<tr>
<td>Y</td>
<td>70</td>
</tr>
<tr>
<td>S</td>
<td>0000 0001 70</td>
</tr>
<tr>
<td>P</td>
<td>70</td>
</tr>
<tr>
<td>PC</td>
<td>15 0</td>
</tr>
</tbody>
</table>
Load Value Into Register (6502)

LDA #$xx
LDA $xxxx

A := xx
A := ($xxxx)^
Load Value Into Accumulator

LDA #$xx
LDA $xxxx
LDA $xxxx,X
LDA $xxxx,Y
LDA ($xx,X)
LDA ($xx),Y

A := xx
A := ($xxxx)^n
A := ($xxxx + X)^n
A := ($xxxx + Y)^n
A := (^word($00xx + X))^n
A := (^word($00xx))^n + Y

Load Value Into Register

LDA #$xx
LDA $xxxx
LDA $xxxx,X
LDA $xxxx,Y
LDX imm/addr
LDY imm/addr

A := xx
A := ($xxxx)^
A := ($xxxx + X)^*
A := ($xxxx + Y)^*

X := imm/addr
Y := imm/addr

2 different variants implies 2 different OPCODEs for LDX!
2 different variants implies additional 2 different OPCODEs for LDY!
& Store Value From Register

LDA #$xx
LDA $xxxx
LDA $xxxx,X
LDA $xxxx,Y
LDX imm/addr
LDY imm/addr
STA $xxxx
STA $xxxx,X
STA $xxxx,Y
STX addr
STY addr

A := xx
A := ($xxxx)^
A := ($xxxx + X)^
A := ($xxxx + Y)^
X := imm/addr
Y := imm/addr
($xxxx)^ := A
($xxxx + X)^ := A
($xxxx + Y)^ := A
addr := X
addr := Y
Move (Transfer) Value Between Registers

LDA #$xx
LDA $xxxx
LDA $xxxx,X
LDA $xxxx,Y
LDX imm/addr
LDY imm/addr
STA $xxxx
STA $xxxx,X
STA $xxxx,Y
STX addr
STY addr

A := xx
A := ($xxxx)^
A := ($xxxx + X)^
A := ($xxxx + Y)^
X := imm/addr
Y := imm/addr
($xxxx)^ := A
($xxxx + X)^ := A
($xxxx + Y)^ := A
addr := X
addr := Y

TAX
TXA
TAY
TYA
TSX
TXS

X := A
A := X
Y := A
A := Y
X := S
S := X
### Push To Stack & Pop (Pull) From Stack

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDA #$xx</td>
<td>A := xx</td>
</tr>
<tr>
<td>LDA $xxxx</td>
<td>A := ($xxxx)^</td>
</tr>
<tr>
<td>LDA $xxxx,X</td>
<td>A := ($xxxx + X)^</td>
</tr>
<tr>
<td>LDA $xxxx,Y</td>
<td>A := ($xxxx + Y)^</td>
</tr>
<tr>
<td>LDX imm/addr</td>
<td>X := imm/addr</td>
</tr>
<tr>
<td>LDY imm/addr</td>
<td>Y := imm/addr</td>
</tr>
<tr>
<td>STA $xxxx</td>
<td>($xxxx)^ := A</td>
</tr>
<tr>
<td>STA $xxxx,X</td>
<td>($xxxx + X)^ := A</td>
</tr>
<tr>
<td>STA $xxxx,Y</td>
<td>($xxxx + Y)^ := A</td>
</tr>
<tr>
<td>STX addr</td>
<td>addr := X</td>
</tr>
<tr>
<td>STY addr</td>
<td>addr := Y</td>
</tr>
<tr>
<td>TAX</td>
<td>X := A</td>
</tr>
<tr>
<td>TXA</td>
<td>A := X</td>
</tr>
<tr>
<td>TAY</td>
<td>Y := A</td>
</tr>
<tr>
<td>TYA</td>
<td>A := Y</td>
</tr>
<tr>
<td>TSX</td>
<td>X := S</td>
</tr>
<tr>
<td>TXS</td>
<td>S := X</td>
</tr>
<tr>
<td>PHP</td>
<td>push P (flags)</td>
</tr>
<tr>
<td>PLP</td>
<td>pop P (flags)</td>
</tr>
<tr>
<td>PHA</td>
<td>push A</td>
</tr>
<tr>
<td>PLA</td>
<td>pop A</td>
</tr>
</tbody>
</table>
# Setting Flags

- **LDA #$xx**
  - A := xx

- **LDA $xxxx**
  - A := ($xxxx)^

- **LDA $xxxx,X**
  - A := ($xxxx + X)^

- **LDA $xxxx,Y**
  - A := ($xxxx + Y)^

- **LDX imm/addr**
  - X := imm/addr

- **LDY imm/addr**
  - Y := imm/addr

- **STA $xxxx**
  - ($xxxx)^ := A
  - ($xxxx)^ := A
  - addr := X
  - addr := Y

- **STA $xxxx,X**

- **STA $xxxx,Y**

- **STX addr**

- **STY addr**

### Flags

- **TAX**
  - X := A

- **TXA**
  - A := X

- **TAY**
  - Y := A

- **TYA**
  - A := Y

- **TSX**
  - X := S

- **TXS**
  - S := X

- **PHP**
  - push P (flags)

- **PLP**
  - pop P (flags)

- **PHA**

- **PLA**
  - pop A

- **P.Negative := target.7**
  - if target = 0 then
    - P.Zero := 1
  - else
    - P.Zero := 0;
### Setting Flags

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDA #$xx</td>
<td>A := xx</td>
</tr>
<tr>
<td>LDA $xxxx</td>
<td>A := ($xxxx)^</td>
</tr>
<tr>
<td>LDA $xxxx,X</td>
<td>A := ($xxxx + X)^</td>
</tr>
<tr>
<td>LDA $xxxx,Y</td>
<td>A := ($xxxx + Y)^</td>
</tr>
<tr>
<td>LDX imm/addr</td>
<td>X := imm/addr</td>
</tr>
<tr>
<td>LDY imm/addr</td>
<td>Y := imm/addr</td>
</tr>
<tr>
<td>STA $xxxx</td>
<td>($xxxx)^ := A</td>
</tr>
<tr>
<td>STA $xxxx,X</td>
<td>($xxxx + X)^ := A</td>
</tr>
<tr>
<td>STA $xxxx,Y</td>
<td>($xxxx + Y)^ := A</td>
</tr>
<tr>
<td>STX addr</td>
<td>addr := X</td>
</tr>
<tr>
<td>STY addr</td>
<td>addr := Y</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAX</td>
<td>X := A</td>
</tr>
<tr>
<td>TXA</td>
<td>A := X</td>
</tr>
<tr>
<td>TAY</td>
<td>Y := A</td>
</tr>
<tr>
<td>TYA</td>
<td>A := Y</td>
</tr>
<tr>
<td>TSX</td>
<td>X := S</td>
</tr>
<tr>
<td>TXS</td>
<td>S := X</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PHP</td>
<td>push P (flags)</td>
</tr>
<tr>
<td>PLP</td>
<td>pop P (flags)</td>
</tr>
<tr>
<td>PHA</td>
<td>push A</td>
</tr>
<tr>
<td>PLA</td>
<td>pop A</td>
</tr>
</tbody>
</table>

```plaintext
P.Negative := target.7
if target = 0 then
  P.Zero := 1
else
  P.Zero := 0;
```

### Flags
- **P.Negative**: target.7
- **P.Zero**: 1 if target = 0, 0 otherwise
- **P.Carry**: 0 or 1
- **P**: push P (flags)
- **S**: push S (flags)
- **X**: push X (flags)
- **Y**: push Y (flags)

![Image](attachment:image.png)
### Bitwise Operations

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA imm/addr</td>
<td>A := A BitwiseOr imm/addr</td>
</tr>
<tr>
<td>AND imm/addr</td>
<td>A := A BitwiseAnd imm/addr</td>
</tr>
<tr>
<td>EOR imm/addr</td>
<td>A := A BitwiseXor imm/addr</td>
</tr>
<tr>
<td>? NOT</td>
<td>EOR #$FF</td>
</tr>
<tr>
<td>ASL A</td>
<td>A := A shl 1</td>
</tr>
<tr>
<td>LSR A</td>
<td>A := A shr 1</td>
</tr>
<tr>
<td>ROL A</td>
<td>A := A rol 1</td>
</tr>
<tr>
<td>ROR A</td>
<td>A := A ror 1</td>
</tr>
</tbody>
</table>

P.Negative := A.7
if A = 0 then
  P.Zero := 1
else
  P.Zero := 0;
**Oring 16-bit Numbers (e.g. Little Endian)**

<table>
<thead>
<tr>
<th>MSB of A stored at $A001</th>
<th>LSB of A stored at $A000</th>
</tr>
</thead>
<tbody>
<tr>
<td>A15 A14 A13 A12 A11 A10 A9 A8</td>
<td>A7 A6 A5 A4 A3 A2 A1 A0</td>
</tr>
</tbody>
</table>

\[ A_{15} A_{14} A_{13} A_{12} A_{11} A_{10} A_{9} A_{8} \]

\[ \text{or} \]

<table>
<thead>
<tr>
<th>MSB of B stored at $B001</th>
<th>LSB of B stored at $B000</th>
</tr>
</thead>
<tbody>
<tr>
<td>B15 B14 B13 B12 B11 B10 B9 B8</td>
<td>B7 B6 B5 B4 B3 B2 B1 B0</td>
</tr>
</tbody>
</table>

\[ B_{15} B_{14} B_{13} B_{12} B_{11} B_{10} B_{9} B_{8} \]

\[ = \]

<table>
<thead>
<tr>
<th>MSB of C stored at $C001</th>
<th>LSB of C stored at $C000</th>
</tr>
</thead>
<tbody>
<tr>
<td>C15 C14 C13 C12 C11 C10 C9 C8</td>
<td>C7 C6 C5 C4 C3 C2 C1 C0</td>
</tr>
</tbody>
</table>

\[ C_{15} C_{14} C_{13} C_{12} C_{11} C_{10} C_{9} C_{8} \]
Oring 16-bit Numbers (e.g. Little Endian)

MSB of A stored at $A001

A15 A14 A13 A12 A11 A10 A9 A8

MSB of B stored at $B001

B15 B14 B13 B12 B11 B10 B9 B8

MSB of C stored at $C001

C15 C14 C13 C12 C11 C10 C9 C8

LSB of A stored at $A000

A7 A6 A5 A4 A3 A2 A1 A0

LSB of B stored at $B000

B7 B6 B5 B4 B3 B2 B1 B0

LSB of C stored at $C000

C7 C6 C5 C4 C3 C2 C1 C0

LDA $A000
ORA $B000
STA $C000

LDA $A001
ORA $B001
STA $C001