Examples of Moodle randomized calculated exercises

In the document, we show the original task texts, scope of the variables, solutions formula, example task realization displayed to a student, and the solution.

# Binary arithmetic

## Binary-decimal conversion

**Task**: The decimal value of the value in NBC {a}{b}{c}{d}{e}{f}{g}{h} is:

**Variables**: a-h binary

**Formula**: {a}\*128+{b}\*64+{c}\*32+{d}\*16+{e}\*8+{f}\*4+{g}\*2+{h}

**Example**: The decimal value of the value in NBC 10000011 is:

**Answer**: 131

### variant – 4221 code divisibility

**Task**: In code 4221 (digit weights), specify whether the number {a}{b}{c}{d} is divisible by {e} (decimal divisor).

0-not, 1-yes

**Variables**: a-d binary, e 2-7

1-ceil(fmod({a}\*4+{b}\*2+{c}\*2+{d},{e})/10)

In code 4221 (digit weights), specify whether the number 0001 is divisible by 6 (decimal divisor).

0-not, 1-yes

**Answer**: 0

## Hexadecimal value

**Task**: The hexadecimal number #{x}B{y} in decimal is:

**Variables**: x,y 0-9

**Formula**: {x}\*256+11\*16+{y}

The hexadecimal number #4B1 in decimal is:

**Answer**: 1201

## Binary fraction

**Task**: The fractional binary number 0.{a}{b}{=1-{a}} is equal to decimal:

**Variables**: a,b binary

**Formula**: 0.5\*{a}+0.25\*{b}+0.125\*(1-{a})

**Example**: The fractional binary number 0.001 is equal to decimal:

**Answer**: 0.125

## Binary fraction as proper decimal fraction

**Task**: The fractional binary number 0.{a}{b}1 is equal to A/B proper decimal fraction (specify as A.B, for example, 5/12, specify as 5.12, do not provide a decimal fraction 0.xxxx):

**Variables**: a,b binary

**Formula**: 4\*{a}+2\*{b}+1+0.8

**Example**: The fractional binary number 0.011 is equal to A/B proper decimal fraction (specify as A.B, for example, 5/12, specify as 5.12, do not provide a decimal fraction 0.xxxx):

**Answer**: 3.8

## Decimal value of Sign-Magnitude binary number given hexadecimal

**Task**: The decimal value of the value in the code Sign-Magnitude, given in hex: #F{e} is:

**Variables**: Variables: e 0-9

**Formula**: -(1\*64+1\*32+1\*16+{e})

**Example**: The decimal value of the value in the code Sign-Magnitude, given in hex: #F4 is:

**Answer**: -116

## Decimal value of U1 binary number given hexadecimal

**Task**: The decimal value of the value in U1 given in hex #A{e} is:

**Variables:** e 0-9

**Formula**: 1\*(-127)+0\*64+1\*32+0\*16+{e}

**Example**: The decimal value of the value in U1 given in hex #A6 is:

**Answer**: -89

### variant – binary input data

**Task**: The decimal value of the byte in the code Sign-Magnitude 11{c}{d}{e}{f}{g}{h} is:

**Variables**: c-h binary

**Formula**: -(1\*64+{c}\*32+{d}\*16+{e}\*8+{f}\*4+{g}\*2+{h})

**Example**: The decimal value of the byte in the code Sign-Magnitude 11111101 is:

**Answer**: -125

## Number of bits needed to represent a set of values

**Task**: A set is the encoded features of some object with values {b}-{a}. At least how many bits are needed to represent the elements of this set?

**Variables**: a,b integer, b<a

**Formula**: ceil(log({a}-{b}+1)/log(2))

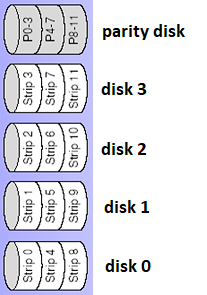
**Example**: A set is the encoded features of some object with values 7-40. At least how many bits are needed to represent the elements of this set?

**Answer**: 6

# Bit operations

## Parity in RAID reconstruction

**Task**: In RAID 4, data is placed in Strips that are "scattered" over the data disks, so that each subsequent strip is on the next data disk, modulo the number of disks. For this, there is a parity disk that holds the parity bits of zeroth bits, first bits, second bits, etc., equal-numbered strips divided by the number of data disks, for example, strips 0-3, 4-7, 8-11, etc.:



The start of strips 0,1,2,3 looks like this:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 0 | 0 | Parity disk |
| 1 | 1 | 1 | 1 | 1 | Disk 3 |
| 1 | 0 | 0 | 1 | 0 | Disk 2 |
| 1 | {a} | {=1-{a}} | {b} | {=1-{b}} | Disk 1 |
| 0 | 1 | 1 | 0 | 0 | Disk 0 |

In the parity strip, the values are placed so that the parity bit keeps the corresponding strip bits 0-3 even.

Disk 3 has been corrupted and always reads 1. After replacing the disk with a new one, what values should I put in the strip on disk 3?

**Variables**: a,b binary

**Formula**: 10000+{a}\*1000+(1-{a})\*100+(1-{b})\*10+(1-{b})

**Example**: In RAID 4, data is placed in Strips that are "scattered" over the data disks, so that each subsequent strip is on the next data disk, modulo the number of disks. For this, there is a parity disk that holds the parity bits of zeroth bits, first bits, second bits, etc., equal-numbered strips divided by the number of data disks, for example, strips 0-3, 4-7, 8-11, etc.:

[the figure]

The start of strips 0,1,2,3 looks like this:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 0 | 0 | Parity disk |
| 1 | 1 | 1 | 1 | 1 | Disk 3 |
| 1 | 0 | 0 | 1 | 0 | Disk 2 |
| 1 | 0 | 1 | 1 | 0 | Disk 1 |
| 0 | 1 | 1 | 0 | 0 | Disk 0 |

In the parity strip, the values are placed so that the parity bit keeps the corresponding strip bits 0-3 even.

Disk 3 has been corrupted and always reads 1. After replacing the disk with a new one, what values should I put in the strip on disk 3?

**Answer**: 10100

## Bit encoding of set elements

**Task**: For the bit-coding of the elements of the set (the universe and its sets represented as binary words), assume the universe to be the set of all decimal digits. Show the representation of the set of digits occurring in the number 9{a}{b}{c}{d}{e} in a 10-bit binary word.

Elements of the universe should be coded according to the value of digits, starting with lsb. The lsb bit is on the right.

**Variables**: a 0-1, b 2-3, c 4-5, d 6-7, e 8-9

**Formula**: 1000000000+pow(10,{a})+pow(10,{b})+pow(10,{c})+pow(10,{d})+pow(10,{e})

**Example**: For the bit-coding of the elements of the set (the universe and its sets represented as binary words), assume the universe to be the set of all decimal digits. Show the representation of the set of digits occurring in the number 902578 in a 10-bit binary word.

Elements of the universe should be coded according to the value of digits, starting with lsb. The lsb bit is on the right.

**Answer**: 1110100101

### Variant – union of sets

**Task**: For the bit-coding of the elements of the set (the universe and its sets represented as binary words), assume the universe to be the set of all decimal digits. For two sets of digits of certain decimal numbers (each set of one number):

9{a}4{c}{d}8

{b}{a}8{d}4

give the encoding of the union of sets in a 10-bit binary word.

Elements of the universe should be coded according to the value of digits, starting with lsb. The lsb bit is on the right.

**Variables**: a 0-1, b 2-3, c 4-5, d fixed 7

**Formula**: 1000000000+pow(10,{a})+pow(10,{b})+pow(10,{c})+pow(10,{d})+pow(10,4)+pow(10,8)

**Example**: For the bit-coding of the elements of the set (the universe and its sets represented as binary words), assume the universe to be the set of all decimal digits. For two sets of digits of certain decimal numbers (each set of one number):

904578

30874

give the encoding of the union of sets in a 10-bit binary word.

Elements of the universe should be coded according to the value of digits, starting with lsb. The lsb bit is on the right.

**Answer**: 1110111001

## Big endian encoding

**Task**: A 32 bit word of hexadecimal value #33445566 was stored at the address {={a}\*4} (decimal) in byte-addressed memory of a Big-Endian computer. What is the address of a byte containing value #{={b}+3}{={b}+3}?

**Variables**: a 1000-2000, b 0-3

**Formula**: {a}\*4+{b}

**Example**: A 32 bit word of hexadecimal value #12345678 was stored at the address 4112 (decimal) in memory of a Big-Endian computer with 8-bit bytes. What is the address of a byte containing value #34?

**Answer**: 4113

## Little endian encoding

**Task**: Specify in hexadecimal format (sequence of hexadecimal digits, but without preceding #) the contents of the bytes (lowest address byte to highest address byte) for little-endian double-word aligned numeric value in NBC binary:

0{c}{=1-{c}}{e} {d}{=1-{d}}0{e} {a}0{=1-{a}}{b} 0{c}{d}{=1-{b}}

We assume the word to have 16 bits.

**Variables**: a-e binary

**Formula**: ({a}\*8000+(1-{a})\*2000+{b}\*1000+{c}\*400+{d}\*200+(1-{b})\*100+{c}\*40+(1-{c})\*20+{e}\*10+{d}\*8+(1-{d})\*4+{e}\*1)\*10000

**Example**: Specify in hexadecimal format (sequence of hexadecimal digits, but without preceding #) the contents of the bytes (lowest address byte to highest address byte) for little-endian double-word aligned numeric value in NBC binary:

0010 0100 0010 0001

We assume the word to have 16 bits.

**Answer**: 21240000

### variant – multiple choice

**Task**: Specify in hexadecimal format (sequence of hexadecimal digits, but without preceding #) the contents of the bytes (lowest address byte to highest address byte) for big-endian double-word aligned numeric value in NBC binary:

0{c}{=1-{c}}{e} {d}{=1-{d}}0{e} {a}0{=1-{a}}{b} 0{c}{d}{=1-{b}}

**Variables**: a-e binary

**The formulas of the distractors:**

1. 0000{={c}\*4+(1-{c})\*2+{e}\*1}{={d}\*8+(1-{d})\*4+{e}\*1}{={a}\*8+(1-{a})\*2+{b}\*1}{={c}\*4+{d}\*2+(1-{b})\*1}
2. {={a}\*8+(1-{a})\*2+{b}\*1}{={c}\*4+{d}\*2+(1-{b})\*1}{={c}\*4+(1-{c})\*2+{e}\*1}{={d}\*8+(1-{d})\*4+{e}\*1}0000
3. 0000{={d}\*8+(1-{d})\*4+{e}\*1}{={c}\*4+(1-{c})\*2+{e}\*1}{={c}\*4+{d}\*2+(1-{b})\*1}{={a}\*8+(1-{a})\*2+{b}\*1}
4. {={c}\*4+{d}\*2+(1-{b})\*1}{={a}\*8+(1-{a})\*2+{b}\*1}{={d}\*8+(1-{d})\*4+{e}\*1}{={c}\*4+(1-{c})\*2+{e}\*1}0000

**Example**: Specify in hexadecimal format (sequence of hexadecimal digits, but without preceding #) the contents of the bytes (lowest address byte to highest address byte) for big-endian double-word aligned numeric value in NBC binary:

0101 0101 1000 0101

1. 00005585
2. 85550000
3. 00005558
4. 58550000

**(a is correct)**

# Floating point

## Binary representation of a floating point number

**Task**: Binary value for a floating point number

{=(-1)\*pow(2,1\*8+{b}\*4+0\*2+{e}-7)\*(1+{f}\*0.5+{g}\*0.25+{h}\*0.125)}

in the format 1(sign)-4(exponent)-3(mantissa), is:

(in the solution, remove the spaces separating the number fields)

**Variables**: b,e-h binary

**Formula**: 1\*10000000+1\*1000000+{b}\*100000+0\*10000+{e}\*1000+{f}\*100+{g}\*10+{h}

**Example**: Binary value for a floating point number

-7

in the format 1(sign)-4(exponent)-3(mantissa), is:

(in the solution, remove the spaces separating the number fields)

**Answer**: 11001110

## Changing the format of a floating point number

**Task**: We have a number in IEEE-like 14-bit FLP code (5-bit exponent). Compress this number to 12-bit format (with a 4-bit exponent). s means sign, e - exponent, m - mantissa.

14 bit format:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| b13-s | b12-e | b11-e | b10-e | b9-e | b8-e | b7-m | b6-m | b5-m | b4-m | b3-m | b2-m | b1-m | b0-m |
| 1 | 1 | 0 | {a} | 0 | {c} | {f} | {=1-{b}} | {i} | 0 | 1 | 1 | 1 | 1 |

 in the result, remove the spaces separating the number fields

**Variables**: a,b,c,f,i binary

**Formula**: 100000000000+10000000000+{a}\*1000000000+{c}\*10000000+{f}\*1000000+(1-{b})\*100000+{i}\*10000+1000

**Example**: We have a number in IEEE-like 14-bit FLP code (5-bit exponent). Compress this number to 12-bit format (with a 4-bit exponent). s means sign, e - exponent, m - mantissa.

14 bit format:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| b13-s | b12-e | b11-e | b10-e | b9-e | b8-e | b7-m | b6-m | b5-m | b4-m | b3-m | b2-m | b1-m | b0-m |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |

 in the result, remove the spaces separating the number fields

**Answer**: 110011001000

### variant - normalization

**Task**: The following floating-point number L consists of a four-bit mantissa and a four-bit exponent. This number is not normalized and the mantissa is full form (no bits are hidden).

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| sign |  | hidden bit |  | exponent | | | |  | mantissa | | | |
| 0 | no | 0 | {g} | 1 | {h} | 0 | 0 | 1 | {a} |

Please write the number ( – L\*2) in FLP normalized form. Hide the bit in the resulting number.

(in the solution, remove the spaces separating the number fields)

**Variables**: a,g,h binary

**Formula**: 100000000+{g}\*1000000+{h}\*10000+{a}\*1000

**Example**: The following floating-point number L consists of a four-bit mantissa and a four-bit exponent. This number is not normalized and the mantissa is full form (no bits are hidden).

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| sign |  | hidden bit |  | exponent | | | |  | mantissa | | | |
| 0 | no | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 |

Please write the number ( – L\*2) in FLP normalized form. Hide the bit in the resulting number.

(in the solution, remove the spaces separating the number fields)

**Answer**: 101000000

## Adding floating point numbers

**Task**: Add 2 FLP numbers in the format 1(sign)-4(exponent)-3(mantissa).

1 {q}10{a} {b}0{c}

1 {q}01{a} {=1-{c}}0{d}

In the result, skip spaces separating the number fields

**Variables**: a-d,q binary

**Formula**: 1\*10000000+{q}\*1000000+1\*100000+0\*10000+{a}\*1000+{b}\*100+1\*10+1+{c}\*{d}\*0

**Example**: Add 2 FLP numbers in the format 1(sign)-4(exponent)-3(mantissa).

1 1100 100

1 1010 101

In the result, skip spaces separating the number fields

**Answer**: 11101111

### variant – convert FLP given binary to decimal

**Task**: Decimal value for a floating point number having binary image {a} {b}10{e} {f}{g}{h}

in the format 1(sign)-4(exponent)-3(mantissa), is (provide a decimal value):

**Variables**: a,b,e-h binary

**Formula**: pow(-1,{a})\*pow(2,{b}\*8+1\*4+(0)\*2+{e}-7)\*(1+{f}\*0.5+{g}\*0.25+{h}\*0.125)

**Example**: Decimal value for a floating point number having binary image 0 1100 110

in the format 1(sign)-4(exponent)-3(mantissa), is (provide a decimal value):

Answer: 56.00000000

### variant – convert FLP given hexadecimal to decimal

**Task**: The decimal value for a floating point number given as hexadecimal #{a}EH

in the format 1(sign)-4(exponent)-3(mantissa), is (provide a decimal value):

**Variables**: a 3-9

**Formula**: pow(-1,floor({a}/8))\*pow(2,{a}-floor({a}/8)\*8+1-7)\*(1+1\*0.5+1\*0.25+0\*0.125)

**Example**: The decimal value for a floating point number given as hexadecimal #6EH

in the format 1(sign)-4(exponent)-3(mantissa), is (provide a decimal value):

Answer: 1.750000000

## Comparing floating point numbers – different exponents

**Task**: Without performing floating point operations, determine which of the numbers is greater. FLP numbers are given in hexadecimal. Format: sign-1, exponent-4, mantissa-4

first: #1{c}{a}

second: #1{d}{b}

which number is greater?

**Variables**: a,b 0-9, c 5-9, d 1-4

**The distractor are:**

1. first #1{c}{a}
2. second #1{d}{b}
3. none of #1{c}{a}, #1{d}{b}

**Example**: Without performing floating point operations, determine which of the numbers is greater. FLP numbers are given in hexadecimal. Format: sign-1, exponent-4, mantissa-4

first: #192

second: #129

which number is greater?

1. first #192
2. second #129
3. none of #192, #129

**(correct solution b)**

### FLP comparing variant – equal exponents, different mantissa

**Task**: Without performing floating point operations, determine which of the numbers is greater. FLP numbers are given in hexadecimal. Format: sign-1, exponent-4, mantissa-4

first: #1{c}{a}

second: #1{c}{b}

which number is greater?

**Variables**: c 1-9, a 5-9, b 0-4

**The distractor are:**

1. first #1{c}{a}
2. second #1{c}{b}
3. none of #1{c}{a}, #1{c}{b}

**Example**: Without performing floating point operations, determine which of the numbers is greater. FLP numbers are given in hexadecimal. Format: sign-1, exponent-4, mantissa-4

first: #187

second: #180

which number is greater?

1. first #187
2. second #180
3. none of #187, #180

**(correct solution b)**

### FLP comparing variant – different signs

**Task**: Without performing floating point operations, determine which of the numbers is greater. FLP numbers are given in hexadecimal. Format: sign-1, exponent-4, mantissa-4

first: #1{c}{a}

second: #0{d}{b}

which number is greater?

**Variables**: a,b 0-9, c,d 1-9

**The distractors are:**

1. first #1{c}{a}
2. second #0{d}{b}
3. none of #1{c}{a}, #0{d}{b}

**Example**: Without performing floating point operations, determine which of the numbers is greater. FLP numbers are given in hexadecimal. Format: sign-1, exponent-4, mantissa-4

first: #137

second: #021

which number is greater?

1. first #1{c}{a}
2. second #0{d}{b}
3. none of #1{c}{a}, #0{d}{b}

**(correct answer b)**

### FLP comparing variant – illegal exponent 0

**Task**: Without performing floating point operations, determine which of the numbers is greater. FLP numbers are given in hexadecimal. Format: sign-1, exponent-4, mantissa-4

first: #{s}0{a}

second: #{t}{d}{b}

which number is greater?

**Variables**: s,t 0-1, d 1-9, a,b 0-9

**The distractors are:**

1. first #{s}0{a}
2. second #{t}{d}{b}
3. none of #{s}0{a}, #{t}{d}{b}

**Example**: Without performing floating point operations, determine which of the numbers is greater. FLP numbers are given in hexadecimal. Format: sign-1, exponent-4, mantissa-4

first: #002

second: #157

which number is greater?

1. first #002
2. second #157
3. none of #002, #157

**(correct solution c)**

### FLP variant – precision

**Task**: The IEEE binary32 format is 1(sign) 8(exponent) 23(mantissa).

Enter the result of adding numbers in IEEE754 binary32 format: 2{={FP}+23} and {arg2} , in binary32 format.

Ignore the rounding.

(the symbol ^ means a power, i.e. 22 is written 2^2)

**Variables**: FP 1-5, arg2 1-15, baddev -1-1

**The distractors are:**

1. 2^{={FP}+23} + {= {arg2} - ({arg2} % (2\*\*({FP})))}
2. 2^{={FP}+23} + {= {arg2} - ({arg2} % (2\*\*({FP}-1)))}
3. 2^{={FP}+23} + {= {arg2} - ({arg2} % (2\*\*({FP}+1)))}
4. 2^{={FP}+23} + {= {arg2} - ({arg2} % (2\*\*({FP}+({baddev}\*2))))}

**Example**: The IEEE binary32 format is 1(sign) 8(exponent) 23(mantissa).

Enter the result of adding numbers in IEEE754 binary32 format: 225 and 15 , in binary32 format.

Ignore the rounding.

(the symbol ^ means a power, i.e. 22 is written 2^2)

1. a.2^25 + 12
2. b.2^25 + 14
3. c.2^25 + 15
4. d.2^25 + 8

**(correct solution a)**

# Fixed point

## Fixed point addition - carry bits

**Task**: Give the carry bits from c4 to c0 (in that order) as a result of adding two 4-bit numbers in the U2 code. Think about how many bits are from c4 to c0.

1{={a2}\*10}{={a1}\*10}{={a0}\*10}

1{={b2}\*10}{={b1}\*10}{={b0}\*10}

"incoming" carry {={c0}\*10}

**Variables**: a0-c0 binary

**Formula**: ceil({c0}\*10\*ceil(0.1+0.1)\*ceil({a2}+{b2})\*ceil({a1}+{b1})\*ceil({a0}+{b0})\*0.1+{a0}\*10\*{b0}\*10\*ceil(0.1+0.1)\*ceil({a2}+{b2})\*ceil({a1}+{b1})\*0.1+{a1}\*10\*{b1}\*10\*ceil(0.1+0.1)\*ceil({a2}+{b2})\*0.1+{a2}\*10\*{b2}\*10\*ceil(0.1+0.1)\*0.1+1\*1\*0.1)\*10000 +ceil({c0}\*10\*ceil({a2}+{b2})\*ceil({a1}+{b1})\*ceil({a0}+{b0})\*0.1+{a0}\*10\*{b0}\*10\*ceil({a2}+{b2})\*ceil({a1}+{b1})\*0.1+{a1}\*10\*{b1}\*10\*ceil({a2}+{b2})\*0.1+{a2}\*10\*{b2}\*10\*0.1)\*1000+ceil({c0}\*10\*ceil({a1}+{b1})\*ceil({a0}+{b0})\*0.1+{a0}\*10\*{b0}\*10\*ceil({a1}+{b1})\*0.1+{a1}\*10\*{b1}\*10\*0.1)\*100 +ceil({c0}\*10\*ceil({a0}+{b0})\*0.1+{a0}\*10\*{b0}\*10\*0.1)\*10 +({c0}\*10)\*1

**Example**: Give the carry bits from c4 to c0 (in that order) as a result of adding two 4-bit numbers in the U2 code. Think about how many bits are from c4 to c0.

1111

1110

"incoming" carry 1

**Answer**: 11111

## Fixed point addition - condition bits

**Task**: Give the result in decimal and the NZVC conditions as a result of adding two 4-bit numbers in the U2 code

00{c}{d}

10{g}{h}

"incoming" carry {i}

Separate the resulting number from the conditions with a point: x.NZVC

**Variables**: binary

**Formula**: -8+{c}\*2+{d}+{g}\*2+{h}+{i}-0.1000

**Example**: Give the result in decimal and the NZVC conditions as a result of adding two 4-bit numbers in the U2 code

0001

1011

"incoming" carry 1

Separate the resulting number from the conditions with a point: x.NZVC

**Answer**: -3.1000

## Arithmetic shift

**Task**: Enter the result of the ASL (arithmetic left) shift of the 4-bit register:

{a}1{c}{d}

Before shifting conditions C={e}, V={f}.

Enter the result as xxxx.CV where xxxx is the image of the shifted register and C and V are the condition bits after shift (carry and overflow)

**Variables**: binary

**Formula**: 1\*1000+{c}\*100+{d}\*10+0+{a}\*0.1+{e}-{e}+{f}-{f}+(1-{a})\*0.01

**Example**: Enter the result of the ASL (arithmetic left) shift of the 4-bit register:

0100

Before shifting conditions C=0, V=0.

Enter the result as xxxx.CV where xxxx is the image of the shifted register and C and V are the condition bits after shift (carry and overflow)

**Answer**: 1000.01

### variant – type of shift

**Task**: The 4-bit register and CV conditions before shifting to the left are as follows

11{c}{d} 0 {f}

After shifting

1{c}{d}0 0 0

What could this shift be?

**Variables**: binary

**The distractors are:**

1. ASL by {=1+{f}\*{c}\*{d}\*0}
2. LSL by {=1+{f}\*{c}\*{d}\*0}
3. CSL by {=1+{f}\*{c}\*{d}\*0}
4. CcSL by {=1+{f}\*{c}\*{d}\*0}

**Example**: The 4-bit register and CV conditions before shifting to the left are as follows

1100 0 0

After shifting

1000 0 0

What could this shift be?

1. b.ASL by 1
2. c.LSL by 1
3. a.CSL by 1
4. d.CcSL by 1

**(a and b are correct)**

## Cascading shift

**Task**: Give the result of the cascading arithmetic shift (ASL - more significant word arithmetic shift left and CsSL - least significant word cyclic left shift via condition bit C) of two 4-bit words (left word is more significant):

11{b}{c} {d}{e}{f}{g}

Before shifting condition C=0.

Think about which word should be shifted first.

Enter the result as xxxx.yyyy where xxxx is the more significant word after the shift.

**Variables**: binary

**Formula**: 1\*1000+{b}\*100+{c}\*10+{d}\*1+{e}\*0.1+{f}\*0.01+{g}\*0.001

**Example**: Give the result of the cascading arithmetic shift (ASL - more significant word arithmetic shift left and CsSL - least significant word cyclic left shift via condition bit C) of two 4-bit words (left word is more significant):

1110 1011

Before shifting condition C=0.

Think about which word should be shifted first.

Enter the result as xxxx.yyyy where xxxx is the more significant word after the shift.

**Answer**: 1101.0110

# Unicode

## UTF-32 – bit sequence exceeding the code

**Task**: For a binary code of {a} bits, what is the maximum number of bytes to represent it in UTF-32?

If impossible, enter -1

**Variables**: a>32

**Formula**: -1-{a}\*0}

**Example**: For a binary code of 40 bits, what is the maximum number of bytes to represent it in UTF-32?

If impossible, enter -1

**Answer**: -1

### variant UTF-32 – maximum length

**Task**: For a binary code of {a} bits, what is the maximum number of bytes to represent it in UTF-32?

If impossible, enter -1

**Variables**: a<33

**Formula**: 4-{a}\*0

**Example**: For a binary code of 7 bits, what is the maximum number of bytes to represent it in UTF-32?

If impossible, enter -1

**Answer**: 4

## UTF-8 – bit sequence exceeding the code

**Task**: Present in UTF-8 the 32-bit code given below (in the solution, the byte order from MSB on the left):

100{=1-{c}}00{b}0 00{a}00{c}00 {=1-{a}}000{b}000 0{a}00{c}00{=1-{b}}

In the solution, separate bytes with a space.

If impossible - enter -1

**Variables**: a-c binary

**Formula**: -1+{a}\*{b}\*{c}\*0

**Example**: Present in UTF-8 the 32-bit code given below (in the solution, the byte order from MSB on the left):

10010000 00100000 00000000 01000001

In the solution, separate bytes with a space.

If impossible - enter -1

**Answer**: -1

### variant UTF-8 – alternative code

**Task**: For UTF-8 code:

01{b}{c}{d}{=1-{c}}{=1-{d}}{a}

specify the shortest alternative UTF-8 code, in which two leading zeros (00) are obligatory in the information part of the code before the first 1 instead of one leading zero. Separate the solution bytes with a point.

**Variables**: binary

**Formula**: 11000000+1+0.1+{b}\*0.001+{c}\*0.0001+{d}\*0.00001+(1-{c})\*0.000001+(1-{d})\*0.0000001+{a}\*0.00000001

**Example**: For UTF-8 code:

01110011

specify the shortest alternative UTF-8 code, in which two leading zeros (00) are obligatory in the information part of the code before the first 1 instead of one leading zero. Separate the solution bytes with a point.

**Answer**: 11000001.10110011

## Number of bytes in UTF-8 encoding

**Task**: For a binary code of {a} bits, what is the minimum number of bytes to represent it in UTF-8?

If impossible, enter -1

**Variables**: a 8-31

**Formula**: floor({a}/5.1+0.8)

**Example**: For a binary code of 11 bits, what is the minimum number of bytes to represent it in UTF-8?

If impossible, enter -1

**Answer**: 2

## Ambiguous UTF-8 encoding

**Task**: For code in UTF-8:

11100000 100{=1-{b}}{c}{b}{=1-{a}}{d} 10{b}{c}{d}{=1-{c}}{=1-{d}}{a}

specify the shortest alternative UTF-8 code. Separate the solution bytes with a point.

**Variables**: a-d binary

**Formula**: 11000000+(1-{b})\*10000+{c}\*1000+{b}\*100+(1-{a})\*10+{d}+0.1+{b}\*0.001+{c}\*0.0001+{d}\*0.00001+(1-{c})\*0.000001+(1-{d})\*0.0000001+{a}\*0.00000001

**Example**: For code in UTF-8:

11100000 10010000 10000111

specify the shortest alternative UTF-8 code. Separate the solution bytes with a point.

**Answer**: 11010000.10000111

# Synthesis

## Filling the Karnaugh table

**Task**: 3-bit sequence xyz is treated as a binary NBC value, for example 000 (x=0, y=0, z=0) is decimal 0. Please fill the Karnaugh table for the function giving 1 for the following decimal values: 0, {b}, {c}, {d} The variable x is msb. Provide the contents of the table in rows, i.e, the bits ABCDEFGH, for example, if C=1 and D=1, give the result 00110000

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Karnaugh table  q = f(x, y, z) | | | | |
|  |  | variables x, y | | | |
|  |  | 00 | 01 | 11 | 10 |
| variable z | 0 | A | B | C | D |
| 1 | E | F | G | H |

**Variables**: b 2-3, c 4-5, d 6-7

**Formula**: 10000000+pow(10,7-(5-{b}))+pow(10,7-{c})+pow(10,7-(13-{d}))

**Example**: 3-bit sequence xyz is treated as a binary NBC value, for example 000 (x=0, y=0, z=0) is decimal 0. Please fill the Karnaugh table for the function giving 1 for the following decimal values: 0, 2, 5, 6 The variable x is msb. Provide the contents of the table in rows, i.e, the bits ABCDEFGH, for example, if C=1 and D=1, give the result 00110000

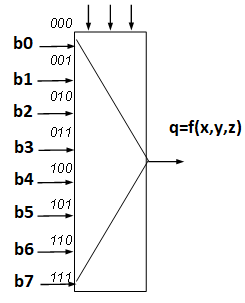
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Karnaugh table  q = f(x, y, z) | | | | |
|  |  | variables x, y | | | |
|  |  | 00 | 01 | 11 | 10 |
| variable z | 0 | A | B | C | D |
| 1 | E | F | G | H |

**Answer**: 10010101

## Multiplexer data input values

**Task**: For the function given in the Kanaugh table, give the values of the successive bits from b0 to b7, which should be inserted into the subsequent information inputs of the multiplexer. The variable x is msb.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Karnaugh table  q = f(x, y, z) | | | | |
|  |  | variables x, y | | | |
|  |  | 00 | 01 | 11 | 10 |
| variable z | 0 | 1 | {b} | {d} | {c} |
| 1 | {a} | {b} | {=1-{d}} | {c} |



**Variables**: a-d binary

**Formula**: 10000000+{a}\*1000000+{b}\*100000+{b}\*10000+{c}\*1000+{c}\*100+{d}\*10+(1-{d})

**Example**: For the function given in the Kanaugh table, give the values of the successive bits from b0 to b7, which should be inserted into the subsequent information inputs of the multiplexer. The variable x is msb.

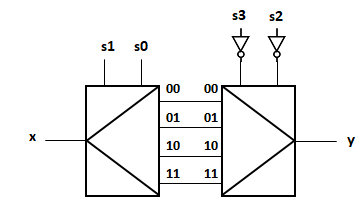
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Karnaugh table  q = f(x, y, z) | | | | |
|  |  | variables x, y | | | |
|  |  | 00 | 01 | 11 | 10 |
| variable z | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 |

**Answer**: 10000001

## Multiplexer control input values

**Task**: What should be applied to the inputs s3, s2 of the system in order for the x signal to be "passed" to the y output, if the multiplexer inputs s1, s0 receive signals {a}{b}? s3 and s1 are msb

**Variables**: binary



**The distractors are:**

1. {=1-{a}}{=1-{b}
2. {=1-{a}}{b}
3. {a}{=1-{b}}
4. {a}{b}

**Example**: What should be applied to the inputs s3, s2 of the system in order for the x signal to be "passed" to the y output, if the multiplexer inputs s1, s0 receive signals 10? s3 and s1 are msb

1. 01
2. 00
3. 11
4. 10

**(a is correct)**

# Addressing

## Addressing mode – register+direct

**Task**: For the given content of registers and RAM, give the value of the instruction argument (for example in LOAD operation). All registers and memory locations are 16-bit. The opcode is 3-bit followed by the addressing mode code (2b) and the register number, the rest is the direct operand in U2. The addressing mode code means:

00 – register mode + direct argument

01 - relative mode + displacement

10 - relative mode with displacement + indirect

11 - relative mode with displacement + double indirect

Values in the tables are given in hexadecimal.

Enter the result as a decimal number.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| register | R0 | R1 | R2 | R3 | R4 | R5 | R6 | R7 |
| content | #09 | #0A | #0B | #0C | #0D | #0E | #0F | #10 |

IR content: #A{={y}\*4+{x}\*2+{z}}F{=8+{v}}

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| RAM - address | #00 | #01 | #02 | #03 | #04 | #05 | #06 | #07 | #08 | #09 | #0A | #0B | #0C | #0D | #0E | #0F |
| content | #0F | #0E | #0D | #0C | #0B | #0A | #09 | #08 | #07 | #06 | #05 | #04 | #03 | #02 | #01 | #00 |

**Variables**: binary

**Formula**: ({y}\*4+{x}\*2+{z}+9-8+{v})

**Example**: For the given content of registers and RAM, give the value of the instruction argument (for example in LOAD operation). All registers and memory locations are 16-bit. The opcode is 3-bit followed by the addressing mode code (2b) and the register number, the rest is the direct operand in U2. The addressing mode code means:

00 – register mode + direct argument

01 - relative mode + displacement

10 - relative mode with displacement + indirect

11 - relative mode with displacement + double indirect

Values in the tables are given in hexadecimal.

Enter the result as a decimal number.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| register | R0 | R1 | R2 | R3 | R4 | R5 | R6 | R7 |
| content | #09 | #0A | #0B | #0C | #0D | #0E | #0F | #10 |

IR content: #A3F9

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| RAM - address | #00 | #01 | #02 | #03 | #04 | #05 | #06 | #07 | #08 | #09 | #0A | #0B | #0C | #0D | #0E | #0F |
| content | #0F | #0E | #0D | #0C | #0B | #0A | #09 | #08 | #07 | #06 | #05 | #04 | #03 | #02 | #01 | #00 |

**Answer**: 5

## Addressing mode – relative+double indirect

**Task**: For the given content of registers and RAM, give the value of the instruction argument. All registers and memory locations are 16-bit. The opcode is 3-bit followed by the addressing mode code (2b) and the register number, the rest is the direct operand in U2. The addressing mode code means:

11 – register mode + direct argument

10 - relative mode + displacement

01 - relative mode with displacement + indirect

00 - relative mode with displacement + double indirect

Values in the tables are given in hexadecimal.

Enter the result as a decimal number.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| register | R0 | R1 | R2 | R3 | R4 | R5 | R6 | R7 |
| content | #09 | #0A | #0B | #0C | #0D | #0E | #0F | #10 |

IR content: #A{={y}\*4+{x}\*2+{z}}F{=8+{v}}

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| RAM - address | #00 | #01 | #02 | #03 | #04 | #05 | #06 | #07 | #08 | #09 | #0A | #0B | #0C | #0D | #0E | #0F |
| content | #02 | #01 | #00 | #0F | #0E | #0D | #0C | #0B | #0A | #09 | #08 | #07 | #06 | #05 | #04 | #03 |

**Variables**: binary

**Formula**: fmod(15-fmod(15-fmod(15-({y}\*4+{x}\*2+{z}+9-8+{v})+3,16)+3,16)+3,16)

**Example**: For the given content of registers and RAM, give the value of the instruction argument. All registers and memory locations are 16-bit. The opcode is 3-bit followed by the addressing mode code (2b) and the register number, the rest is the direct operand in U2. The addressing mode code means:

11 – register mode + direct argument

10 - relative mode + displacement

01 - relative mode with displacement + indirect

00 - relative mode with displacement + double indirect

Values in the tables are given in hexadecimal.

Enter the result as a decimal number.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| register | R0 | R1 | R2 | R3 | R4 | R5 | R6 | R7 |
| content | #09 | #0A | #0B | #0C | #0D | #0E | #0F | #10 |

IR content: #A6F9

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| RAM - address | #00 | #01 | #02 | #03 | #04 | #05 | #06 | #07 | #08 | #09 | #0A | #0B | #0C | #0D | #0E | #0F |
| content | #02 | #01 | #00 | #0F | #0E | #0D | #0C | #0B | #0A | #09 | #08 | #07 | #06 | #05 | #04 | #03 |

**Answer**: 10

### variant – index mode with automodification

**Task**: For the given content of registers and RAM, give the value of the instruction argument (for example in operation LOAD). All registers and memory locations are 16-bit. The opcode is 3 bits, then the addressing mode code (2b) and the register number, then the scale (2 bits) and the index register number (only the first 4 registers), the rest is the direct argument in U2. The addressing mode code means:

11 – relative + index mode with scale and postdecrement + displacement

10 – relative + index mode with sale and predecrement + displacement

01 - relative + index mode with scale and postincrement + displacement

00 - relative + index mode with scale and preincrement + displacement

The scale is given as a power of 2.

Values in the tables are given in hexadecimal.

Specify in decimal the value of the effective operand of the instruction, and after point - the content of the index register after execution of the instruction, also in decimal as a two-digit number. Example: 35.36

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| register | R0 | R1 | R2 | R3 | R4 | R5 | R6 | R7 |
| content | #08 | #09 | #0A | #0B | #0C | #0D | #0E | #0F |

IR content: #D{={y}\*4+{x}\*2+{z}}{={v}\*4+{y}\*2+(1-{z})}{=8+{x}}

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| RAM - address | #00 | #01 | #02 | #03 | #04 | #05 | #06 | #07 | #08 | #09 | #0A | #0B | #0C | #0D | #0E | #0F |
| content | #1F | #1E | #1D | #1C | #1B | #1A | #19 | #18 | #17 | #16 | #15 | #14 | #13 | #12 | #11 | #10 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| RAM - address | #10 | #11 | #12 | #13 | #14 | #15 | #16 | #17 | #18 | #19 | #1A | #1B | #1C | #1D | #1E | #1F |
| content | #0F | #0E | #0D | #0C | #0B | #0A | #09 | #08 | #07 | #06 | #05 | #04 | #03 | #02 | #01 | #00 |

**Variables**: binary

**Formula**: 31-({y}\*4+{x}\*2+{z}+8+({y}\*2+(1-{z})+8-1)\*pow(2,{v})-8+{x})+0.01\*(8+{y}\*2+(1-{z})-1)

**Example**: For the given content of registers and RAM, give the value of the instruction argument. All registers and memory locations are 16-bit. The opcode is 3 bits, then the addressing mode code (2b) and the register number, then the scale (2 bits) and the index register number (only the first 4 registers), the rest is the direct argument in U2. The addressing mode code means:

11 – relative + index mode with scale with postdecrement + displacement

10 – relative + index mode with scale with predecrement + displacement

01 - relative + index mode with scale with postincrement + displacement

00 - relative + index mode with scale with preincrement + displacement

The scale is given as a power of 2.

Values in the tables are given in hexadecimal - the value is preceded by #.

Specify the value of the instruction's effective operand in decimal integer, and the contents of the index register after execution of the instruction in decimal places after point.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| register | R0 | R1 | R2 | R3 | R4 | R5 | R6 | R7 |
| content | #1 | #2 | #3 | #4 | #5 | #6 | #7 | #8 |

content of IR: #B100

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| RAM - address | #00 | #01 | #02 | #03 | #04 | #05 | #06 | #07 | #08 | #09 | #0A | #0B | #0C | #0D | #0E | #0F |
| content | #1F | #1E | #1D | #1C | #1B | #1A | #19 | #18 | #17 | #16 | #15 | #14 | #13 | #12 | #11 | #10 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| RAM - address | #10 | #11 | #12 | #13 | #14 | #15 | #16 | #17 | #18 | #19 | #1A | #1B | #1C | #1D | #1E | #1F |
| content | #0F | #0E | #0D | #0C | #0B | #0A | #09 | #08 | #07 | #06 | #05 | #04 | #03 | #02 | #01 | #00 |

**Answer**: 29.0

# Stack

## Performing CALL instruction on the stack and PC register

**Task**: In the CALL instruction (subroutine call) indicated by the PC register, the jump address is relative (relative to the PC) by {a}. The stack is empty ascending.

Give the new values of the SP register after executing the CALL instruction and the value stored on the stack while executing this instruction. Values should be separated by a point: SP.value saved.

All values are in decimal.

|  |  |
| --- | --- |
| REGISTERS | content before |
| PC  (*currentPC* value) | {c} |
| SP | {b} |

**Variables**: a 5-15, c 7350-7359, b 2340-2349

**Formula**: ({b}+1)+({c}+1)\*0.0001+{a}\*0

**Example**: In the CALL instruction (subroutine call) indicated by the PC register, the jump address is relative (relative to the PC) by 7. The stack is empty ascending.

Give the new values of the SP register after executing the CALL instruction and the value stored on the stack while executing this instruction. Values should be separated by a point: SP.value saved.

All values are in decimal.

|  |  |
| --- | --- |
| REGISTERS | content before |
| PC | 7351 |
| SP | 2348 |

Answer: 2349.7352

## Stack pointer initialization

**Task**: A program running on CPU with an "empty ascending" stack in byte-addressable memory should use the range of memory addresses from {=({initsp} - 1)\*1000} to {={initsp}\*1000 - 1} (decimal) as the stack . Before starting the program, Stack Pointer register should be initialized to:

**Variables**: initsp 150-250

**Formula**: ({initsp} - 1)\*1000

**Example**: A program running on CPU with an "empty ascending" stack in byte-addressable memory should use the range of memory addresses from 199000 to 199999 (decimal) as the stack . Before starting the program, Stack Pointer register should be initialized to:

**Answer**: 199000

## Stack frame elements – dynamic link location

**Task**: The stack is empty ascending. The frame pointer is used. Addressing is 16-bit. Memory cells and registers are 16-bit. The subroutine has local variables: {a}, which occupy 1 memory location each. After calling the subroutine with parameters: {b}, occupying 1 memory location each, after executing the prologue, the dynamic link will be at the address relative to the SP (decimal number should be entered):

**Variables**: a,b 1-4

**Formula**: -({a}+1)+{b}-{b}

The stack is empty ascending. The frame pointer is used. Addressing is 16-bit. Memory cells and registers are 16-bit. The subroutine has local variables: 2, which occupy 1 memory location each. After calling the subroutine with parameters: 4, occupying 1 memory location each, after executing the prologue, the dynamic link will be at the address relative to the SP (decimal number should be entered):

**Answer**: -3

## Stack frame elements – parameter location

**Task**: The stack is full descending. The frame pointer is used. Addressing is 16-bit. Memory cells and registers are 16-bit. The subroutine has local variables: {a}, which occupy 1 memory location each. After calling the subroutine with parameters: {b}, occupying 1 memory location each, after executing the prologue, the parameter {=floor({b}/2)+1} will be at the address relative to the SP (specify a decimal number):

**Variables**: a,b 1-4

**Formula**: ({a}+2+floor({b}/2))

**Example**: The stack is full descending. The frame pointer is used. Addressing is 16-bit. Memory cells and registers are 16-bit. The subroutine has local variables: 2, which occupy 1 memory location each. After calling the subroutine with parameters: 4, occupying 1 memory location each, after executing the prologue, the parameter 3 will be at the address relative to the SP (specify a decimal number):

**Answer**: 6

### variant – stack pointer value

**Task**: The stack is empty descending. The frame pointer is used. Addressing is 16-bit. Memory cells and registers are 16-bit. The stack pointer is #{x}4{y}{b}. The subroutine has 2 local variables occupying 1 memory location each. After calling the subroutine with 1 parameter occupying 1 memory location, after executing the prologue, the stack pointer will be equal to (specify a hexadecimal number but without #):

**Variables**: x 1-9, y 0-9, b 0-5

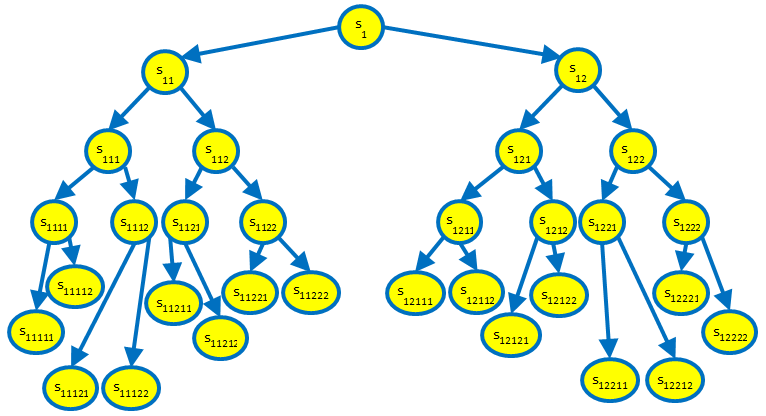
**Formula**: {x}\*1000+400+{y}\*10+{b}-5

**Example**: The stack is empty descending. The frame pointer is used. Addressing is 16-bit. Memory cells and registers are 16-bit. The stack pointer is #8488. The subroutine has 2 local variables occupying 1 memory location each. After calling the subroutine with 1 parameter occupying 1 memory location, after executing the prologue, the stack pointer will be equal to (specify a hexadecimal number but without #):

**Answer**: 8483

## Dynamic link – call of parent subroutine

**Task**: The figure shows the static nesting structure of subroutines. Subroutine s{=floor((10000+{a1}\*1000+{b1}\*100+{c1}\*10+{d1})/pow(10,{e1}))} makes a call to the subroutine that is its "parent" in the hierarchy. To what stack frame (of which subroutine) will the dynamic link be made in the frame of the called subroutine?

****Enter the name of the subroutine with the index, without the letter "s", e.g. for s11122 enter 11122

**Variables**: a1-d1 1-2, e1 0-3

**Formula**: floor((10000+{a1}\*1000+{b1}\*100+{c1}\*10+{d1})/pow(10,{e1}))

**Example**: The figure shows the static nesting structure of subroutines. Subroutine s121 makes a call to the subroutine that is its "parent" in the hierarchy. To what stack frame (of which subroutine) will the dynamic link be made in the frame of the called subroutine?

Enter the name of the subroutine with the index, without the letter "s", e.g. for s11122 enter 11122

**Answer**: 121

## Dynamic link

**Task**: The figure shows the static nesting of subroutines. In the stack frame of the subroutine s{=floor((10000+{a1}\*1000+{b1}\*100+{c1}\*10+{d1})/pow(10,{e1}))}, can there be a dynamic link to the frame of the subroutine s{=floor((10000+{a1}\*1000+{b2}\*100+{c1}\*10+{d1})/pow(10,{e2}))}?

answer 0-not, 1-yes

[the same figure as above]

**Variables**: a1-d2 1-2, e1 2-3, e2 0-2

**Formula**: 1-ceil((abs({e2}-1-{e1})/100)+abs(floor(floor((10000+{a1}\*1000+{b1}\*100+{c1}\*10+{d1})/pow(10,{e1}))/10)-floor((10000+{a1}\*1000+{b2}\*100+{c1}\*10+{d1})/pow(10,{e2})))/100000)+{b2}\*{b2}\*{d2}\*0

**Example**: The figure shows the static nesting of subroutines. In the stack frame of the subroutine s11 can there be a dynamic link to the frame of the subroutine s11221?

answer 0-not, 1-yes

**Answer**: 1

### variant

The figure shows the static nesting of subroutines. In the stack frame of the subroutine s{=floor((10000+{a1}\*1000+{b1}\*100+{c1}\*10+{d1})/pow(10,{e1}))}, can there be a dynamic link to the frame of the subroutine s{=floor((10000+{a1}\*1000+{b2}\*100+{c1}\*10+{d1})/pow(10,{e2}))}?

answer 0-not, 1-yes

[the same figure as above]

**Variables**: a1-d2 1-2, e1 0-1, e2 2

**Formula**: 1-ceil((abs({e2}-1-{e1})/100)+abs(floor(floor((10000+{a1}\*1000+{b1}\*100+{c1}\*10+{d1})/pow(10,{e1}))/10)-floor((10000+{a1}\*1000+{b2}\*100+{c1}\*10+{d1})/pow(10,{e2})))/100000)+{b2}\*{b2}\*{d2}\*0

**Example**: The figure shows the static nesting structure of subroutines. In the stack frame of the subroutine s11111, can there be a dynamic link to the frame of the subroutine s111?

answer 0-not, 1-yes

**Answer**: 0

**ChatGPT** is not tested since the task depends highly on the figure

## Static link

**Task**: The figure shows the static nesting structure of subroutines. Can there be a static link to the frame of the subroutine s{=floor((10000+{a1}\*1000+{b2}\*100+{c1}\*10+{d2})/pow(10,{e2}))} in the frame of the subroutine s{=floor((10000+{a1}\*1000+{b1}\*100+{c1}\*10+{d1})/pow(10,{e1}))}?

answer 0-no, 1-yes

[the same figure as above]

**Variables**: a1-d2 1-2, e1 0-3, e2 1-4

**Formula**: 1-ceil((abs({e2}-1-{e1})/100)+abs(floor(floor((10000+{a1}\*1000+{b1}\*100+{c1}\*10+{d1})/pow(10,{e1}))/10)-floor((10000+{a1}\*1000+{b2}\*100+{c1}\*10+{d2})/pow(10,{e2})))/100000)

**Example**: The figure shows the static nesting structure of subroutines. Can there be a static link to the frame of the subroutine s1 in the frame of the subroutine s1221?

answer 0-no, 1-yes

**Answer**: 0

### variant – static link in recursive call

**Task**: The figure shows the static nesting structure of subroutines. Subroutine s{=floor((10000+{a1}\*1000+{b1}\*100+{c1}\*10+{d1})/pow(10,{e1}))} makes a recursive call. To what stack frame (which subroutine) will the static link be in the subroutine frame called recursively?

Enter the name of the subroutine with the index, without the letter "s", e.g. for s11122 enter 11122

[the same figure as above]

**Variables**: a1-d1 1-2, e1 0-3

**Formula**: floor((10000+{a1}\*1000+{b1}\*100+{c1}\*10+{d1})/pow(10,{e1}+1))

**Example**: The figure shows the static nesting structure of subroutines. Subroutine s122 makes a recursive call. To what stack frame (which subroutine) will the static link be in the subroutine frame called recursively?

Enter the name of the subroutine with the index, without the letter "s", e.g. for s11122 enter 11122

**Answer**: 12

# Paging

## Paging example

**Task**: The virtual address consists of 8b page number and 8b offset. The page index table is shown below (index, content). For decimal address {=({s3}\*4+{s2}\*2+{s1})\*256+({i7}\*128+{i5}\*32+{i4}\*16+{i1}\*2)}, binary 0000 0{s3}{s2}{s1} {i7}0{i5}{i4} 00{i1}0, enter the physical address in the form: frame number.offset (as decimal numbers, offset in 3 digits ). For example, for a physical address consisting of frame 0 and offset 18, specify 0.018. If there is no physical address for the given virtual address, then -1 should be specified.

|  |  |
| --- | --- |
| 7 | {=fmod(pow(2, 7)+1,13)\*(1-({i4})\*({i1}))-(({i4})\*({i1}))} |
| 6 | {=fmod(pow(2, 6)+1,13)\*(1-({i4})\*(1-{i1}))-(({i4})\*(1-{i1}))} |
| 5 | {=fmod(pow(2, 5)+1,13)\*(1-(1-{i4})\*({i1}))-((1-{i4})\*({i1}))} |
| 4 | {=fmod(pow(2, 4)+1,13)\*(1-(1-{i4})\*(1-{i1}))-((1-{i4})\*(1-{i1}))} |
| 3 | {=fmod(pow(2, 3)+1,13)\*(1-({i4})\*({i1}))-(({i4})\*({i1}))} |
| 2 | {=fmod(pow(2, 2)+1,13)\*(1-({i4})\*(1-{i1}))-(({i4})\*(1-{i1}))} |
| 1 | {=fmod(pow(2, 1)+1,13)\*(1-(1-{i4})\*({i1}))-((1-{i4})\*({i1}))} |
| 0 | {=fmod(pow(2, 0)+1,13)\*(1-(1-{i4})\*(1-{i1}))-((1-{i4})\*(1-{i1}))} |

**Variables**: binary

**Formula**: (fmod(pow(2,{s3}\*4+{s2}\*2+{s1})+1,13)+({i7}\*128+{i5}\*32+{i4}\*16+{i1}\*2)/1000)\*(floor(abs({s2}\*2+{s1}-{i4}\*2-{i1})/4+0.9))-(1-floor(abs({s2}\*2+{s1}-{i4}\*2-{i1})/4+0.9))

**Example**: The virtual address consists of 8b page number and 8b offset. The page index table is shown below (index, content). For decimal address 1840, binary 0000 0111 0011 0000, enter the physical address in the form: frame number.offset (as decimal numbers, offset in 3 digits ). For example, for a physical address consisting of frame 0 and offset 18, specify 0.018. If there is no physical address for the given virtual address, then -1 should be specified.

|  |  |
| --- | --- |
| 7 | 12 |
| 6 | -1 |
| 5 | 7 |
| 4 | 4 |
| 3 | 9 |
| 2 | -1 |
| 1 | 3 |
| 0 | 2 |

**Answer**: 12.048

## Paging address space – physical memory address space

**Task**: Assuming that memory cells are 1-byte, the page number in the address field is {a} bits, the offset is {b} bits, the frame number is {c} bits, and all entries in the TIS page index table are on a {=pow(2,{d})}-byte word boundary, please specify:

- the maximum size of the program's physical memory in MB

**Variables**: binary

**Formula**: pow(2,{b}-10+{c}-10)+{a}\*{d}\*0

**Example**: Assuming that memory cells are 1-byte, the page number in the address field is 12 bits, the offset is 12 bits, the frame number is 11 bits, and all entries in the TIS page index table are on a 4-byte word boundary, please specify:

- the maximum size of the program's physical memory in MiB

**Answer**: 8

### variant - virtual memory address space

**Task**: Assuming that memory cells are 1-byte, the page number in the address field is {a} bits, the offset is {b} bits, the frame number is {c} bits, and all entries in the PIT page index table are on a {=pow(2,{d})}-byte word boundary, specify:

- TIS maximum size of the program in kB

**Variables**: a 10-13, d 1-3, b 10-13, c 10-13

**Formula**: pow(2,{a}-10+{d})+{b}-{b}+{c}-{c}

**Example**: Assuming that memory cells are 1-byte, the page number in the address field is 13 bits, the offset is 13 bits, the frame number is 11 bits, and all entries in the PIT page index table are on a 4-byte word boundary, please specify:

- the maximum size of the program's physical memory in MiB

**Answer**: 16

### variant – PIT size

**Task**: Assuming that memory cells are 1-byte, the page number in the address field is {a} bits, the offset is {b} bits, the frame number is {c} bits, and all entries in the PIT page index table are on a {=pow(2,{d})}-byte word boundary, specify:

- PIT maximum size of the program in kiB

**Variables**: a 10-13, b 10-13, c 10-13, d 1-3

**Formula**: pow(2,{a}+{d}-10)+{b}\*{c}\*0

**Example**: Assuming that memory cells are 1-byte, the page number in the address field is 13 bits, the offset is 12 bits, the frame number is 13 bits, and all entries in the PIT page index table are on a 4-byte word boundary, specify:

- PIT maximum size of the program in kiB

**Answer**: 32

# Cache

## Associative cache line size

**Task**: Memory has {=pow(2,{b})} Mi (mebi - binary equivalent of mega) cells. The cache is associative and has {=pow(2,{a})}ki lines (kibi - the binary equivalent of kilo). The tag is {=20+{b}-{d}} bits. How many cells of size {=8\*pow(2,{e})} bits are in a cache line?

**Variables**: a 1-3, b 0-4, d 1-3, e 0-3

**Formula**: pow(2,{d})+{a}\*{b}\*{e}\*0

**Example**: Memory has 2 Mi (mebi - binary equivalent of mega) cells. The cache is associative and has 8ki lines (kibi - the binary equivalent of kilo). The tag is 19 bits. How many cells of size 16 bits are in a cache line?

**Answer**: 4

### variant – tag size

**Task**: Memory has {=pow(2,{b})} Mi (mebi - binary equivalent of mega) cells. The cache is associative and has {=pow(2,{a})}ki lines (kibi - the binary equivalent of kilo). There are {=pow(2,{d})} cells of size {=8\*pow(2,{e})} bits in a cache line. How many bits is a tag?

**Variables**: a 1-3, b 0-4, d 1-3, e 0-3

**Formula**: (20+{b}-{d})+{a}\*{e}\*0

**Example**: Memory has 8 Mi (mebi - binary equivalent of mega) cells. The cache is associative and has 4ki lines (kibi - the binary equivalent of kilo). There are 8 cells of size 32 bits in a cache line. How many bits is a tag?

**Answer**: 20

### variant – address space

**Task**: Cache is associative, it has {=pow(2,{b})} ki (kibi - binary equivalent of kilo) lines. Tags are {=19+{c}} bits. The line has {=pow(2,{d})} cells of size {=8\*pow(2,{e})} bits. What is the memory address space (in Mi cells)?

**Variables**: b 0-4, c 0-4, d 1-3, e 0-3,

**Formula**: pow(2,{c}-1+{d})+{b}\*{e}\*0

**Example**: Cache is associative, it has 4 ki (kibi - binary equivalent of kilo) lines. Tags are 23 bits. The line has 4 cells of size 32 bits. What is the memory address space (in Mi cells)?

**Answer**: 32

## Directly addressed cache line size

**Task**: Memory has {=pow(2,{b})} Mi (mebi - binary equivalent of mega) cells. The cache is directly addressable and has {=pow(2,7+{b}-{c}-{d})}ki lines (kibi - the binary equivalent of kilo). Tags are {={c}+3} bits. The cells are {=8\*pow(2,{e})} bits. How many cells are there in a line?

**Variables**: b 0-3, c 0-4, d 1-3, e 0-3

**Formula**: pow(2,{d})+{b}\*{c}\*{e}\*0

**Example**: Memory has 2 Mi (mebi - binary equivalent of mega) cells. The cache is directly addressable and has 64 ki lines (kibi - the binary equivalent of kilo). Tags are 4 bits. The cells are 64 bits. How many cells are there in a line?

**Answer**: 2

### variant – tag size

**Task**: Memory has {=pow(2,{b})} Mi (mebi - binary equivalent of mega) cells. The cache is directly addressable and has {=pow(2,4+{b}-{c}-{d})}ki lines (kibi - the binary equivalent of kilo). The lines have {=pow(2,{d})} cells of {=8\*pow(2,{e})} bits. How many bits are tags?

**Variables**: b 0-3, c 0-4, d 1-3, e 0-3

**Formula**: ({c}+6)+{b}\*{d}\*{e}\*0

**Example**: Memory has 1 Mi (mebi - binary equivalent of mega) cells. The cache is directly addressable and has 4ki lines (kibi - the binary equivalent of kilo). The lines have 4 cells of 16 bits. How many bits are tags?

**Answer**: 6

### variant – lines in the cache

**Task**: Memory has {=pow(2,{b})} Mi (mebi - binary equivalent of mega) cells. The cache is directly addressable. The tags are {=10+{c}} bits. The lines have {=pow(2,{d})} cells of {=8\*pow(2,{e})} bits. How many ki lines are there in the cache?

**Variables**: b 9-10, c 0-4, d 1-3, e 0-3

**Formula**: pow(2,{b}-{c}-{d})+{e}\*0

**Example**: Memory has 1024 Mi (mebi - binary equivalent of mega) cells. The cache is directly addressable.

The tags are 11 bits. The lines have 2 cells of 32 bits. How many ki lines are there in the cache?

**Answer**: 256

### variant – memory size

**Task**: The cache is directly addressable, it has {=pow(2,{b})} ki (kibi - binary equivalent of mega) lines. The tags are {=10+{c}} bits. The lines have {=pow(2,{d})} cells of {=8\*pow(2,{e})} bits. What is the memory address space (in Mi cells, mebi - binary equivalent of mega)?

**Variables**: b 0-4, c 0-4, d 1-3, e 0-3

**Formula**: pow(2,{b}+{c}+{d})+{e}\*0

**Example**: The cache is directly addressable, it has 8 ki (kibi - binary equivalent of mega) lines. The tags are 13 bits. The lines have 8 cells of 32 bits. What is the memory address space (in Mi cells, mebi - binary equivalent of mega)?

**Answer**: 512

## Skewed set-associative cache block size

**Task**: The skewed set-associative cache has the number of blocks: {=pow(2,{a})}. The memory has {=pow(2,{b})} Mi (mebi - binary equivalent of mega) cells. The tags are {=10+{c}} bits. In a line of each block there are {=pow(2,{d})} cells of size {=8\*pow(2,{e})} bits. How many ki rows (kibi - the binary equivalent of kilo) does each cache block have?

**Variables**: a 1-3, b 9-10, c 0-4, d 1-3, e 0-3

**Formula**: pow(2,{b}-{c}-{d})+{a}\*{e}\*0

**Example**: The skewed set-associative cache has the number of blocks: 4. The memory has 512 Mi (mebi - binary equivalent of mega) cells. The tags are 14 bits. In a line of each block there are 4 cells of size 16 bits. How many ki rows (kibi - the binary equivalent of kilo) does each cache block have?

**Answer**:

### variant – number of cells in a line

**Task**: The skewed set-associative cache has the number of blocks: {=pow(2,{a})}, and lines in each block: {=pow(2,1+{b}-{c}-{d})}ki (kibi - the binary equivalent of kilo). The memory has {=pow(2,{b}-6)} Mi (mebi - binary equivalent of mega) cells. The tags are {=3+{c}} bits. How many cells does each cache line in a block have?

**Variables**: a 1-3, b 9-10, c 0-4, d 1-3, e 0-3

**Formula**: pow(2,{d})+{a}\*{b}\*{c}\*{e}\*0

**Example**: The skewed set-associative cache has the number of blocks: 4, and lines in eah block: 64 ki (kibi - the binary equivalent of kilo). The memory has 16 Mi (mebi - binary equivalent of mega) cells. The tags are 5 bits. How many cells does each cache line in a block have?

**Answer**: 8

### variant – tag size

**Task**: The skewed set-associative cache has the number of blocks: {=pow(2,{a})}, and lines in each block: {=pow(2,{b}-2-{c}-{d})} ki (kibi - the binary equivalent of kilo). The memory has {=pow(2,{b}-6)} Mi (mebi - binary equivalent of mega) cells. In a line of a block there are {=pow(2,{d})} cells of {=8\*pow(2,{e})} bits. How many bits are tags?

**Variables**: a 1-3, b 9-10, c 0-4, d 1-3, e 0-3

**Formula**: ({c}+6)+{a}\*{b}\*{d}\*{e}\*0

**Example**: The skewed set-associative cache has the number of blocks: 4, and lines in each block: 16 ki (kibi - the binary equivalent of kilo). The memory has 16 Mi (mebi - binary equivalent of mega) cells. In a line of a block there are 2 cells of 32 bits. How many bits are tags?

**Answer**: 9

### variant – memory space size

**Task**: The skewed set-associative cache has the number of blocks: {=pow(2,{a})}, and lines in each block: {=pow(2,{b})} ki (kibi - the binary equivalent of kilo). The tags are {=10+{c}} bits. In a line of a block there are {=pow(2,{d})} cells of {=8\*pow(2,{e})} bits. What is the memory address space (in Mi cells, mebi - binary equivalent of mega)?

**Variables**: a 1-3, b 0-4, c 0-4, d 1-3, e 0-3

**Formula**: pow(2,{b}+{c}+{d})+{a}\*{e}\*0

**Example**: The skewed set-associative cache has the number of blocks: 2, and lines in each block: 2 ki (kibi - the binary equivalent of kilo). The tags are 14 bits. In a line of a block there 4 cells of 16 bits. What is the memory address space (in Mi cells, mebi - binary equivalent of mega)?

**Answer**: 128

## Line address in directly addressed cache

**Task**: Byte-addressed memory has a 16-bit address. A directly addressed cache has 7-bit tags and 8-byte lines.

Specify the cache line address if the memory address is hexadecimal #{=8000+{x}\*1000+{y}\*400+200+{x}\*40+{y}\*10+{z}\*8+{v4}\*4+{v2}\*2+{v1}}. The result should be given in hexadecimal, omitting the # character.

**Variables**: binary

**Formula**: 80+{x}\*10+{y}\*2+{z}+{v4}\*{v2}\*{v1}\*0

**Example**: Byte memory has a 16-bit address. A directly addressed cache has 7-bit tags and 8-byte lines. Specify the cache line address if the memory address is hexadecimal #8202. The result should be given in hexadecimal, omitting the # character.

**Answer**: 80

### variant - line address in skewed set-associative cache

**Task**: Byte-addressed memory has a 16-bit address. A skewed set-associative cache has 7-bit tags and 4-byte lines.

For the memory address: hexadecimal #{={x}\*4+{z}\*2+(1-{v})\*1}{={x}\*4+({v})\*2+{z}\*1}{=4+{x}\*2}{=(1-{z})\*4+{v2}\*2+{v1}} give the address of the line in the cache hexadecimal, omitting the # character.

**Variables**: binary

**Formula**: {z}\*40+10+({x})\*8+(1-{z})+{v}\*{v2}\*{v1}\*0

**Example**: Byte-addressed memory has a 16-bit address. A skewed set-associative cache has 7-bit tags and 4-byte lines.

For the memory address: hexadecimal #2340 give the address of the line in the cache hexadecimal, omitting the # character.

**Answer**: 50

## Directly addressed cache response

**Task**: Byte memory has a 16-bit address. A directly addressed cache has 7-bit tags and 8-byte lines.

In the example below, enter the pocket response on a hit or the address of the line to be replaced on a miss. All values are given in hexadecimal and the result should be entered as such, omitting the # character.

Content bytes are addressed from 0 to the left.

Effective address from the processor: #{=8000+{x}\*1000+{y}\*400+{z}\*200+{x}\*100+40+{z}\*20+{y}\*10+{v4}\*4+{v2}\*2+{v1}}

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Address | Tag | Content | | | | | | | |
| #08 | #40 | #00 | #40 | #20 | #60 | #10 | #50 | #30 | #70 |
| #0A | #42 | #02 | #42 | #22 | #62 | #12 | #52 | #32 | #72 |
| #0C | #41 | #01 | #41 | #21 | #61 | #11 | #51 | #31 | #71 |
| #0E | #43 | #03 | #43 | #23 | #63 | #13 | #53 | #33 | #73 |
| ... | ... |  |  |  |  |  |  |  |  |
| #28 | #48 | #04 | #44 | #24 | #64 | #14 | #54 | #34 | #74 |
| #2A | #4A | #06 | #46 | #26 | #66 | #16 | #56 | #36 | #76 |
| #2C | #49 | #05 | #45 | #25 | #65 | #15 | #55 | #35 | #75 |
| #2E | #4B | #07 | #47 | #27 | #67 | #17 | #57 | #37 | #77 |

**Variables**: binary

**Formula**: {v1}\*40+{v2}\*20+{v4}\*10+{x}\*4+{y}\*2+{z}

**Example**: Byte memory has a 16-bit address. A directly addressed cache has 7-bit tags and 8-byte lines.

In the example below, enter the pocket response on a hit or the address of the line to be replaced on a miss. All values are given in hexadecimal and the result should be entered as such, omitting the # character.

Content bytes are addressed from 0 to the left.

Effective address from the processor: #9773

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Address | Tag | Content | | | | | | | |
| #08 | #40 | #00 | #40 | #20 | #60 | #10 | #50 | #30 | #70 |
| #0A | #42 | #02 | #42 | #22 | #62 | #12 | #52 | #32 | #72 |
| #0C | #41 | #01 | #41 | #21 | #61 | #11 | #51 | #31 | #71 |
| #0E | #43 | #03 | #43 | #23 | #63 | #13 | #53 | #33 | #73 |
| ... | ... |  |  |  |  |  |  |  |  |
| #28 | #48 | #04 | #44 | #24 | #64 | #14 | #54 | #34 | #74 |
| #2A | #4A | #06 | #46 | #26 | #66 | #16 | #56 | #36 | #76 |
| #2C | #49 | #05 | #45 | #25 | #65 | #15 | #55 | #35 | #75 |
| #2E | #4B | #07 | #47 | #27 | #67 | #17 | #57 | #37 | #77 |

**Answer**: 67

## Skewed set-associative cache – original address

**Task**: Byte- addressable memory has a 16-bit address. A 2-way skewed set-associative cache has 7-bit tags and 4-byte line.

In the following example, enter the effective address from the processor that caused the cache response #{={x}\*40+{y}\*20+{z}\*10+{v}}. All values are given in hexadecimal and the result should be entered as such, but not preceded by a # character.

Content bytes are addressed from 0 to the left.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Address | 1 | tag | content | | | | 2 | tag | content | | | |
| #20 |  | #40 | #00 | #01 | #02 | #03 |  | #42 | #20 | #21 | #22 | #23 |
| #21 |  | #41 | #10 | #11 | #12 | #13 |  | #43 | #30 | #31 | #32 | #33 |
| ... |  | ... |  |  |  |  |  | ... |  |  |  |  |
| #60 |  | #48 | #40 | #41 | #42 | #43 |  | #4A | #60 | #61 | #62 | #63 |
| #61 |  | #49 | #50 | #51 | #52 | #53 |  | #4B | #70 | #71 | #72 | #73 |

**Variables**: x,y,z binary, v 0-3

**Formula**: 8000+{x}\*1000+{y}\*400+{z}\*200+{x}\*100+80+{z}\*4+{v}

**Example**: Byte-addressable memory has a 16-bit address. A 2-way skewed set-associative cache has 7-bit tags and 4-byte lines.

In the following example, enter the effective address from the processor that caused the cache response #72. All values are given in hexadecimal and the result should be entered as such, but not preceded by a # character.

Content bytes are addressed from 0 to the left.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Address | 1 | tag | content | | | | 2 | tag | content | | | |
| #20 |  | #40 | #00 | #01 | #02 | #03 |  | #42 | #20 | #21 | #22 | #23 |
| #21 |  | #41 | #10 | #11 | #12 | #13 |  | #43 | #30 | #31 | #32 | #33 |
| ... |  | ... |  |  |  |  |  | ... |  |  |  |  |
| #60 |  | #48 | #40 | #41 | #42 | #43 |  | #4A | #60 | #61 | #62 | #63 |
| #61 |  | #49 | #50 | #51 | #52 | #53 |  | #4B | #70 | #71 | #72 | #73 |

**Answer**: 9786

### variant - cache response - hit

**Task**: Byte-addressed memory has a 16-bit address. A skewed 2-way set-associative cache has 7-bit tags and 4-byte lines.

In the example below, enter the cache response on a hit or the address of the row to be replaced on a miss. All values are given in hexadecimal and the result should be entered as such.

Content bytes are addressed from 0 to the left.

Effective address from processor: #{=8000+{x}\*1000+{y}\*400+{z}\*200+{x}\*100+80+{z}\*4+{v}}. All values are given in hexadecimal and the result should be entered as such, but without preceding it with a character #.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Address | 1 | tag | content | | | | 2 | tag | content | | | |
| #20 |  | #40 | #00 | #01 | #02 | #03 |  | #42 | #20 | #21 | #22 | #23 |
| #21 |  | #41 | #10 | #11 | #12 | #13 |  | #43 | #30 | #31 | #32 | #33 |
| ... |  | ... |  |  |  |  |  | ... |  |  |  |  |
| #60 |  | #48 | #40 | #41 | #42 | #43 |  | #4A | #60 | #61 | #62 | #63 |
| #61 |  | #49 | #50 | #51 | #52 | #53 |  | #4B | #70 | #71 | #72 | #73 |

**Variables**: x,y,z binary, v 0-3

**Formula**: {x}\*40+{y}\*20+{z}\*10+{v}

**Example**: Byte-addressed memory has a 16-bit address. A skewed 2-way set-associative cache has 7-bit tags and 4-byte lines.

In the example below, enter the cache response on a hit or the address of the row to be replaced on a miss. All values are given in hexadecimal and the result should be entered as such.

Content bytes are addressed from 0 to the left.

Effective address from processor: #8285. All values are given in hexadecimal and the result should be entered as such, but without preceding it with a character #..

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Address | 1 | tag | content | | | | 2 | tag | content | | | |
| #20 |  | #40 | #00 | #01 | #02 | #03 |  | #42 | #20 | #21 | #22 | #23 |
| #21 |  | #41 | #10 | #11 | #12 | #13 |  | #43 | #30 | #31 | #32 | #33 |
| ... |  | ... |  |  |  |  |  | ... |  |  |  |  |
| #60 |  | #48 | #40 | #41 | #42 | #43 |  | #4A | #60 | #61 | #62 | #63 |
| #61 |  | #49 | #50 | #51 | #52 | #53 |  | #4B | #70 | #71 | #72 | #73 |

**Answer**: 11

### variant - cache response - miss

**Task**: Byte-addressed memory has a 16-bit address. A skewed 2-way set-associative cache has 7-bit tags and 4-byte rows.

In the example below, enter the cache response on a hit or the address of the row to be replaced on a miss. All values are given in hexadecimal and the result should be entered as such.

Content bytes are addressed from 0 to the left.

Effective address from processor: #{=4000+{x}\*1000+{y}\*400+{z}\*200+{x}\*100+80+{z}\*4+{v}}. All values are given in hexadecimal and the result should be entered as such, but not preceded by a # character.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Address | 1 | tag | content | | | | 2 | tag | content | | | |
| #20 |  | #40 | #00 | #01 | #02 | #03 |  | #42 | #20 | #21 | #22 | #23 |
| #21 |  | #41 | #10 | #11 | #12 | #13 |  | #43 | #30 | #31 | #32 | #33 |
| ... |  | ... |  |  |  |  |  | ... |  |  |  |  |
| #60 |  | #48 | #40 | #41 | #42 | #43 |  | #4A | #60 | #61 | #62 | #63 |
| #61 |  | #49 | #50 | #51 | #52 | #53 |  | #4B | #70 | #71 | #72 | #73 |

**Variables**: x,y,z binary, v 0-3

**Formula**: {x}\*40+20+{z}+{y}\*{v}\*0

**Example**: Byte-addressed memory has a 16-bit address. A skewed 2-way set-associative cache has 7-bit tags and 4-byte rows.

In the example below, enter the cache response on a hit or the address of the row to be replaced on a miss. All values are given in hexadecimal and the result should be entered as such.

Content bytes are addressed from 0 to the left.

Effective address from processor: #5581. All values are given in hexadecimal and the result should be entered as such, but not preceded by a # character.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Address | 1 | tag | content | | | | 2 | tag | content | | | |
| #20 |  | #40 | #00 | #01 | #02 | #03 |  | #42 | #20 | #21 | #22 | #23 |
| #21 |  | #41 | #10 | #11 | #12 | #13 |  | #43 | #30 | #31 | #32 | #33 |
| ... |  | ... |  |  |  |  |  | ... |  |  |  |  |
| #60 |  | #48 | #40 | #41 | #42 | #43 |  | #4A | #60 | #61 | #62 | #63 |
| #61 |  | #49 | #50 | #51 | #52 | #53 |  | #4B | #70 | #71 | #72 | #73 |

**Answer**: 60

# Predictors

## Two-level predictor with per-branch adaptation - PHT size

**Task**: In a two-level predictor, with individual adaptation (per-branch) BTB table has {a}-bit address, and BHR has {b} bits. How many rows does the PHT table have?

**Variables**: a 2-4, b 2-5

**Formula**: pow(2,{a})+{b}\*0

**Example**: In a two-level predictor, with individual adaptation (per-branch) BTB table has 3-bit address, and BHR has 3 bits. How many rows does the PHT table have?

**Answer**: 8

### variant - global adaptation

**Task**: In a two-level predictor, with global adaptation, the BHR has {b} bits. How many rows does the PHT table have?

**Variable**: b 2-5

**Formula**: pow(2,{b})

**Example**: In a two-level predictor, with global adaptation, the BHR has 4 bits. How many rows does the PHT table have?

**Answer**: 16

## Two-level predictor with set adaptation - PHT size

**Task**: In a two-level predictor, with set adaptation, the index function has {a} bits and the BHR has {b} bits. How many rows does the BHR table have?

**Variables**: a 2-4, b 2-5

**Formula**: pow(2,{a})+{b}\*0

**Example**: In a two-level predictor, with set adaptation, the index function has 4 bits and the BHR has 5 bits. How many rows does the BHR table have?

**Answer**: 16

### variant – BTB table size for per-branch adaptation

**Task**: For per-branch adaptation, the BHR table has {a} rows and the history registers have {b} bits. How many rows does the BTB table have?

**Variables**: a 2-4, b 2-5

**Formula**: {a}+{b}\*0

**Example**: For per-branch adaptation, the BHR table has 4 rows and the history registers have 5 bits. How many rows does the BTB table have?

**Answer**: 4

## The GAp predictor – prediction and new state of prediction automaton

**Task**: BTB table is:

|  |  |
| --- | --- |
| index | content |
| 0 | #FE2A |
| 1 | #FE3A |
| 2 | #FE6A |
| 3 | #FE7A |
| ... | ... |

..

The address in the instruction counter is #FE{=({j})\*4+2+({k})}A

Predictive automata have 4 coded states SNT-0/WNT-1/WT-2/ST-3.

In the BHR history register, coding is NT-0, T-1

The BHR history register has the form (on the left the youngest entry, which is also the least significant bit of the PHT addressing)

|  |  |  |  |
| --- | --- | --- | --- |
| {={d}} | {={c}} | {={b}} | {={a}} |

PHT table (shown horizontally) SNT-0/WNT-1/WT-2/ST-3

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| adr | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| ind 0 wart | 2 | 3 | 1 | 2 | 1 | 2 | 0 | 1 | 3 | 0 | 2 | 3 | 2 | 3 | 1 | 2 |
| ind 1 wart | 3 | 0 | 2 | 3 | 2 | 3 | 1 | 2 | 0 | 1 | 3 | 0 | 3 | 0 | 2 | 3 |
| ind 2 wart | 0 | 1 | 3 | 0 | 3 | 0 | 2 | 3 | 1 | 2 | 0 | 1 | 0 | 1 | 3 | 0 |
| ind 3 wart | 1 | 2 | 0 | 1 | 0 | 1 | 3 | 0 | 2 | 3 | 1 | 2 | 1 | 2 | 0 | 1 |
| ... |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

What will be the prediction? (NT-0, T-1)

Jump has been taken (T-1 taken/NT-0 not taken): {e}

What will be the new state of the predictor automaton? (SNT-0/WNT-1/WT-2/ST-3)

The result should be given in the form of prediction.state; for example, for T.SNT, the result is 1.0

**Variables**: binary

**Formula**: floor((fmod({j}\*2+{k}+{a}+(1-{b})+(1-{c})+({d}),4))/2)+round((fmod({j}\*2+{k}+{a}+(1-{b})+(1-{c})+({d}),4)+({e}\*2-1)+1)\*0.7-0.25)\*0.1

**Example**: The predictor is GAp

BTB table is:

|  |  |
| --- | --- |
| index | content |
| 0 | #FE2A |
| 1 | #FE3A |
| 2 | #FE6A |
| 3 | #FE7A |
| ... | ... |

..

The address in the instruction counter is #FE3A

Predictive automata have 4 coded states SNT-0/WNT-1/WT-2/ST-3.

In the BHR history register, coding is NT-0, T-1

The BHR history register has the form (on the left the youngest entry, which is also the least significant bit of the PHT addressing)

|  |  |  |  |
| --- | --- | --- | --- |
| {={d}} | {={c}} | {={b}} | {={a}} |

PHT table (shown horizontally) SNT-0/WNT-1/WT-2/ST-3

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| adr | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| ind 0 wart | 2 | 3 | 1 | 2 | 1 | 2 | 0 | 1 | 3 | 0 | 2 | 3 | 2 | 3 | 1 | 2 |
| ind 1 wart | 3 | 0 | 2 | 3 | 2 | 3 | 1 | 2 | 0 | 1 | 3 | 0 | 3 | 0 | 2 | 3 |
| ind 2 wart | 0 | 1 | 3 | 0 | 3 | 0 | 2 | 3 | 1 | 2 | 0 | 1 | 0 | 1 | 3 | 0 |
| ind 3 wart | 1 | 2 | 0 | 1 | 0 | 1 | 3 | 0 | 2 | 3 | 1 | 2 | 1 | 2 | 0 | 1 |
| ... |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

What will be the prediction? (NT-0, T-1)

Jump has been taken (T-1 taken/NT-0 not taken): 0

What will be the new state of the predictor automaton? (SNT-0/WNT-1/WT-2/ST-3)

The result should be given in the form of prediction.state; for example, for T.SNT, the result is 1.0

**Answer**: 1.1

### variant – PAp predictor

**Task**: The predictor is of type PAp

The BTB table has the form:

|  |  |
| --- | --- |
| index | value |
| 0 | #B82C |
| 1 | #B83C |
| 2 | #B86C |
| 3 | #B87C |
| ... | ... |

In the history register, coding NT-0, T-1

The BHR history register has the form (on the left the youngest entry, which is also the least significant bit of the PHT addressing)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| BHR(0) | {={d}\*(1-{q})} | {={c}} | {={b}\*(1-{p})} | {={a}} |
| BHR(1} | {={d}\*({q})} | {={c}} | {={b}\*(1-{p})} | {={a}} |
| BHR(2) | {={d}\*(1-{q})} | {={c}} | {={b}\*({p})} | {={a}} |
| BHR(3) | {={d}\*({q})} | {={c}} | {={b}\*({p})} | {={a}} |

The address in the instruction counter is #B8{=({p})\*4+2+({q})}C

Predictive automata have 4 states coded SNT-0/WNT-1/WT-2/ST-3.

PHT table (shown horizontally) SNT-0/WNT-1/WT-2/ST-3

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| adr | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| ind 0 wart | 1 | 1 | 2 | 2 | 1 | 1 | 2 | 2 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| ind 1 wart | 2 | 1 | 3 | 2 | 2 | 1 | 3 | 2 | 1 | 0 | 2 | 1 | 1 | 0 | 2 | 1 |
| ind 2 wart | 1 | 1 | 2 | 2 | 2 | 2 | 3 | 3 | 0 | 0 | 1 | 1 | 1 | 1 | 2 | 2 |
| ind 3 wart | 2 | 1 | 3 | 2 | 3 | 2 | 0 | 3 | 1 | 0 | 2 | 1 | 2 | 1 | 3 | 2 |

What will be the prediction? (NT-0, T-1)

Jump has been taken (T-1 taken/NT-0 not taken): {e}

What will be the new state of the predictor automaton? (SNT-0/WNT-1/WT-2/ST-3)

The result should be given in the form of prediction,state; for example, for T,SNT, specify 1.0

**Variables**: binary

**Formula**: floor((fmod((1-{a})+({b})\*{p}+({c})+(1-{d})\*({q}),4))/2)+round((fmod((1-{a})+({b})\*{p}+({c})+(1-{d})\*({q}),4)+({e}\*2-1)+1) \*0.7-0.25)\*0.1

**Example**: The predictor is of type PAp

The BTB table has the form:

|  |  |
| --- | --- |
| index | value |
| 0 | #B82C |
| 1 | #B83C |
| 2 | #B86C |
| 3 | #B87C |
| ... | ... |

In the history register, coding NT-0, T-1

The BHR history register has the form (on the left the youngest entry, which is also the least significant bit of the PHT addressing)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| BHR(0) | 0 | 0 | 0 | 1 |
| BHR(1} | 0 | 0 | 0 | 1 |
| BHR(2) | 0 | 0 | 1 | 1 |
| BHR(3) | 0 | 0 | 1 | 1 |

The address in the instruction counter is #B86C

Predictive automata have 4 states coded SNT-0/WNT-1/WT-2/ST-3.

PHT table (shown horizontally) SNT-0/WNT-1/WT-2/ST-3

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| adr | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| ind 0 wart | 1 | 1 | 2 | 2 | 1 | 1 | 2 | 2 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| ind 1 wart | 2 | 1 | 3 | 2 | 2 | 1 | 3 | 2 | 1 | 0 | 2 | 1 | 1 | 0 | 2 | 1 |
| ind 2 wart | 1 | 1 | 2 | 2 | 2 | 2 | 3 | 3 | 0 | 0 | 1 | 1 | 1 | 1 | 2 | 2 |
| ind 3 wart | 2 | 1 | 3 | 2 | 3 | 2 | 0 | 3 | 1 | 0 | 2 | 1 | 2 | 1 | 3 | 2 |

What will be the prediction? (NT-0, T-1)

Jump has been taken (T-1 taken/NT-0 not taken): 1

What will be the new state of the predictor automaton? (SNT-0/WNT-1/WT-2/ST-3)

The result should be given in the form of prediction.state; for example, for T.SNT, specify 1.0

**Answer**: 0.2

### variant – SAs predictor

**Task**: The predictor is of type SAs

The address in the instruction counter is #C{={p}\*4+{j}}{=({q})\*8+(1-{k})\*1}7

The F1 function mapping the address into the index of the BHR table is a combination of the 8th bit in the affirmation and the 4th bit in the affirmation.

The F2 function mapping the address into the index of the PHT table is a combination of the 11th bit in the affirmation and the 7th bit in the negation.

The BHR history register has the form (the youngest entry on the left, which is also the least significant bit of the PHT addressing), in the history register, coding NT-0, T-1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| BHR(F1=0) | {={d}\*(1-{q})} | {={c}} | {={b}\*(1-{p})} | {={a}} |
| BHR(F1=1) | {={d}\*({q})} | {={c}} | {={b}\*(1-{p})} | {={a}} |
| BHR(F1=2) | {={d}\*(1-{q})} | {={c}} | {={b}\*({p})} | {={a}} |
| BHR(F1=3) | {={d}\*({q})} | {={c}} | {={b}\*({p})} | {={a}} |

Predictive automata have 4 states coded SNT-0/WNT-1/WT-2/ST-3.

PHT table (shown horizontally, indexed F2) WNT-0/SNT-1/ST-2/WT-3

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| adr | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| ind 0 wart | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| ind 1 wart | 2 | 1 | 1 | 0 | 2 | 1 | 1 | 0 | 2 | 1 | 1 | 0 | 2 | 1 | 1 | 0 |
| ind 2 wart | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| ind 3 wart | 1 | 0 | 1 | 0 | 2 | 1 | 1 | 1 | 2 | 1 | 2 | 1 | 3 | 2 | 3 | 2 |

What will be the prediction? (NT-0, T-1)

Jump has been taken (T-1 taken/NT-0 not taken): {e}

What will be the new state of the predictor automaton? (SNT-0/WNT-1/WT-2/ST-3)

The result should be given in the form of prediction.state; for example, for T.SNT, specify 1.0

**Variables**: binary

**Formula**: floor((fmod(({a})\*{p}+({b})\*{j}+(1-{c})\*{q}+(1-{d})\*(1-{k}),4))/2)+round((fmod(({a})\*{p}+({b})\*{j}+(1-{c})\*{q}+(1-{d})\*(1-{k}),4)+({e}\*2-1)+1) \*0.7-0.25)\*0.1

**Example**: The predictor is of type SAs

The address in the instruction counter is #C597

The F1 function mapping the address into the index of the BHR table is a combination of the 8th bit in the affirmation and the 4th bit in the affirmation.

The F2 function mapping the address into the index of the PHT table is a combination of the 11th bit in the affirmation and the 7th bit in the negation.

The BHR history register has the form (the youngest entry on the left, which is also the least significant bit of the PHT addressing), in the history register, coding NT-0, T-1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| BHR(F1=0) | 0 | 1 | 0 | 0 |
| BHR(F1=1) | 1 | 1 | 0 | 0 |
| BHR(F1=2) | 0 | 1 | 1 | 0 |
| BHR(F1=3) | 1 | 1 | 1 | 0 |

Predictive automata have 4 states coded SNT-0/WNT-1/WT-2/ST-3.

PHT table (shown horizontally, indexed F2) WNT-0/SNT-1/ST-2/WT-3

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| adr | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| ind 0 wart | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| ind 1 wart | 2 | 1 | 1 | 0 | 2 | 1 | 1 | 0 | 2 | 1 | 1 | 0 | 2 | 1 | 1 | 0 |
| ind 2 wart | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| ind 3 wart | 1 | 0 | 1 | 0 | 2 | 1 | 1 | 1 | 2 | 1 | 2 | 1 | 3 | 2 | 3 | 2 |

What will be the prediction? (NT-0, T-1)

Jump has been taken (T-1 taken/NT-0 not taken): 1

What will be the new state of the predictor automaton? (SNT-0/WNT-1/WT-2/ST-3)

The result should be given in the form of prediction.state; for example, for T.SNT, specify 1.0

**Answer**: 0.2

## The GAs predictor – prediction and new state of BHR

**Task**: The predictor is GAs

The address in the instruction counter is #FE{=({j})\*8+(1-{k})}A

The function mapping the address into the index of the PHT table is a combination of the 7th bit in the affirmation and the 4th bit in the negation.

Predictive automata have 4 states coded SNT-0/WNT-1/WT-2/ST-3.

In the history register, coding NT-0, T-1

The BHR history register has the form (on the left the youngest entry, which is also the least significant bit of the PHT addressing)

|  |  |  |  |
| --- | --- | --- | --- |
| {={d}} | {={c}} | {={b}} | {={a}} |

PHT table (shown horizontally) SNT-0/WNT-1/WT-2/ST-3

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| addr | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| ind 0 value | 2 | 3 | 1 | 2 | 1 | 2 | 0 | 1 | 3 | 0 | 2 | 3 | 2 | 3 | 1 | 2 |
| ind 1 value | 3 | 0 | 2 | 3 | 2 | 3 | 1 | 2 | 0 | 1 | 3 | 0 | 3 | 0 | 2 | 3 |
| ind 2 value | 0 | 1 | 3 | 0 | 3 | 0 | 2 | 3 | 1 | 2 | 0 | 1 | 0 | 1 | 3 | 0 |
| ind 3 value | 1 | 2 | 0 | 1 | 0 | 1 | 3 | 0 | 2 | 3 | 1 | 2 | 1 | 2 | 0 | 1 |

What will be the prediction? (NT-0, T-1)

Jump has been taken (T-1 taken/NT-0 not taken): {e}

What will be the new state of the BHR history register after the instruction is executed? (4 bits starting with the youngest)?

The result should be given in the form of prediction.BHR, for example, for T.1010, enter 1.1010 **Variables**: binary

**Formula:** floor((fmod({j}\*2+{k}+(1-{a})+{b}+(1-{c})+{d},4))/2)+{e}\*0.1+{d}\*0.01+{c}\*0.001+{b}\*0.0001+{a}\*0

**Example**: The predictor is GAs

The address in the instruction counter is #FE0A

The function mapping the address into the index of the PHT table is a combination of the 7th bit in the affirmation and the 4th bit in the negation.

Predictive automata have 4 states coded SNT-0/WNT-1/WT-2/ST-3.

In the history register, coding NT-0, T-1

The BHR history register has the form (on the left the youngest entry, which is also the least significant bit of the PHT addressing)

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 1 | 0 | 1 |

PHT table (shown horizontally) SNT-0/WNT-1/WT-2/ST-3

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| addr | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| ind 0 value | 2 | 3 | 1 | 2 | 1 | 2 | 0 | 1 | 3 | 0 | 2 | 3 | 2 | 3 | 1 | 2 |
| ind 1 value | 3 | 0 | 2 | 3 | 2 | 3 | 1 | 2 | 0 | 1 | 3 | 0 | 3 | 0 | 2 | 3 |
| ind 2 value | 0 | 1 | 3 | 0 | 3 | 0 | 2 | 3 | 1 | 2 | 0 | 1 | 0 | 1 | 3 | 0 |
| ind 3 value | 1 | 2 | 0 | 1 | 0 | 1 | 3 | 0 | 2 | 3 | 1 | 2 | 1 | 2 | 0 | 1 |

What will be the prediction? (NT-0, T-1)

Jump has been taken (T-1 taken/NT-0 not taken): 1

What will be the new state of the BHR history register after the instruction is executed? (4 bits starting with the youngest)?

The result should be given in the form of prediction.BHR, for example, for T.1010, enter 1.1010

**Answer**: 0.1010

### variant – PAp predictor

**Task**: The predictor is of type PAp

The BTB table has the form:

|  |  |
| --- | --- |
| index | value |
| 0 | #B82C |
| 1 | #B83C |
| 2 | #B86C |
| 3 | #B87C |
| ... | ... |

In the history register, coding NT-0, T-1

The BHR history register has the form (on the left the youngest entry, which is also the least significant bit of the PHT addressing)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| BHR(0) | {={d}\*(1-{q})} | {={c}} | {={b}\*(1-{p})} | {={a}} |
| BHR(1} | {={d}\*({q})} | {={c}} | {={b}\*(1-{p})} | {={a}} |
| BHR(2) | {={d}\*(1-{q})} | {={c}} | {={b}\*({p})} | {={a}} |
| BHR(3) | {={d}\*({q})} | {={c}} | {={b}\*({p})} | {={a}} |

The address in the instruction counter is #B8{=({p})\*4+2+({q})}C

Predictive automata have 4 states coded SNT-0/WNT-1/WT-2/ST-3.

PHT table (shown horizontally) SNT-0/WNT-1/WT-2/ST-3

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| addr | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| ind 0 value | 1 | 1 | 2 | 2 | 1 | 1 | 2 | 2 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| ind 1 value | 2 | 1 | 3 | 2 | 2 | 1 | 3 | 2 | 1 | 0 | 2 | 1 | 1 | 0 | 2 | 1 |
| ind 2 value | 1 | 1 | 2 | 2 | 2 | 2 | 3 | 3 | 0 | 0 | 1 | 1 | 1 | 1 | 2 | 2 |
| ind 3 value | 2 | 1 | 3 | 2 | 3 | 2 | 0 | 3 | 1 | 0 | 2 | 1 | 2 | 1 | 3 | 2 |

What will be the prediction? (NT-0, T-1)

Jump has been taken (T-1 taken/NT-0 not taken): {e}

What will be the new state of the BHR history register after the instruction is executed? (4 bits starting with the youngest)

The result should be given in the form of prediction,register, for example, for T,1010, enter 1.1010

**Variables**: binary

**Formula:** floor((fmod((1-{a})+({b})\*{p}+({c})+(1-{d})\*{q},4))/2)+{e}\*0.1+{d}\*0.01+{c}\*0.001+{b}\*0.0001

**Example**: The predictor is of type PAp

The BTB table has the form:

|  |  |
| --- | --- |
| index | value |
| 0 | #B82C |
| 1 | #B83C |
| 2 | #B86C |
| 3 | #B87C |
| ... | ... |

In the history register, coding NT-0, T-1

The BHR history register has the form (on the left the youngest entry, which is also the least significant bit of the PHT addressing)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| BHR(0) | 0 | 0 | 0 | 0 |
| BHR(1} | 0 | 0 | 0 | 0 |
| BHR(2) | 0 | 0 | 1 | 0 |
| BHR(3) | 0 | 0 | 1 | 0 |

The address in the instruction counter is #B87C

Predictive automata have 4 states coded SNT-0/WNT-1/WT-2/ST-3.

PHT table (shown horizontally) SNT-0/WNT-1/WT-2/ST-3

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| addr | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| ind 0 value | 1 | 1 | 2 | 2 | 1 | 1 | 2 | 2 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| ind 1 value | 2 | 1 | 3 | 2 | 2 | 1 | 3 | 2 | 1 | 0 | 2 | 1 | 1 | 0 | 2 | 1 |
| ind 2 value | 1 | 1 | 2 | 2 | 2 | 2 | 3 | 3 | 0 | 0 | 1 | 1 | 1 | 1 | 2 | 2 |
| ind 3 value | 2 | 1 | 3 | 2 | 3 | 2 | 0 | 3 | 1 | 0 | 2 | 1 | 2 | 1 | 3 | 2 |

What will be the prediction? (NT-0, T-1)

Jump has been taken (T-1 taken/NT-0 not taken): 0

What will be the new state of the BHR history register after the instruction is executed? (4 bits starting with the youngest)

The result should be given in the form of prediction.register, for example, for T.1010, enter 1.1010

**Answer**: 1.0001