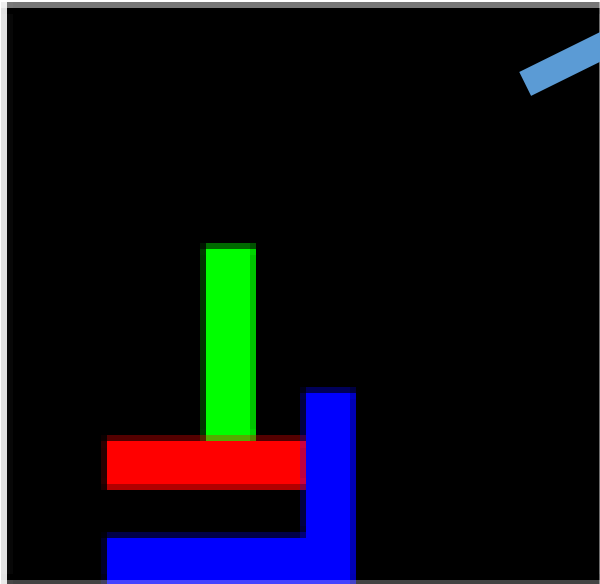
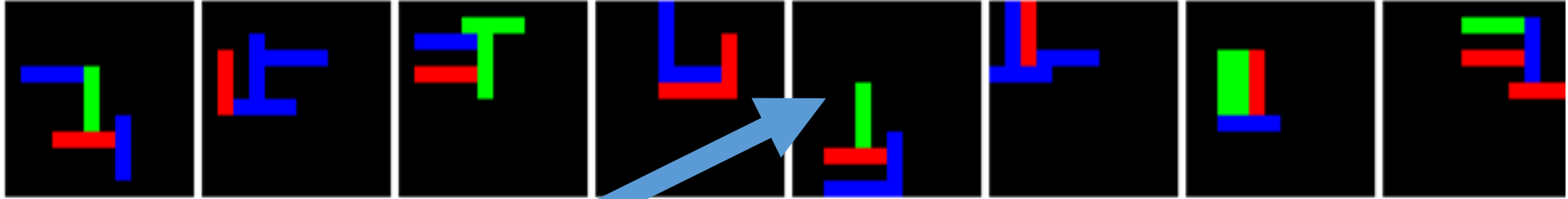


Project 1: Icon Matching

Exact Match Detection Puzzle



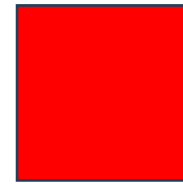
- There will always be eight candidate matches (0-7)
- One pattern must be matched (report a value 0-7)
- There will always be exactly one EXACT match among the candidates
- The parts of the pattern are not necessarily connected
- Do not make assumptions about the number or size of the parts of the pattern.
- The nonblack pixels of the pattern will not occur as a strict subset of the nonblack pixels of any candidate

Icon and Color Palette

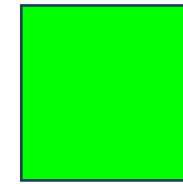
1	1	1	1	3	0	0	0	0	0	0	0
0	0	0	0	3	0	0	0	0	0	0	0
0	0	0	1	3	0	0	0	0	0	0	0
0	0	0	1	3	0	0	0	0	0	0	0
0	0	0	1	2	2	2	2	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	...				



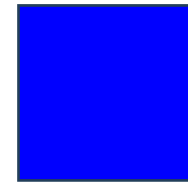
0



1



2



3

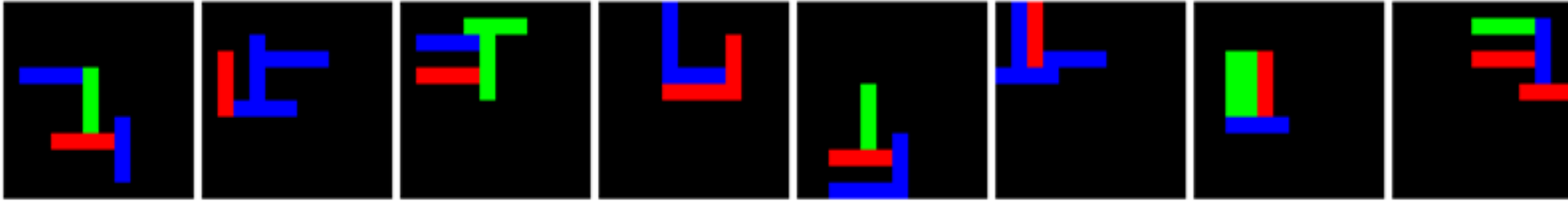
- Each pixel value is 1 of 4 colors
- Each icon is 12×12 pixels
- Background is all black (0)
 - Shown as gray here
- Grid is NOT present in actual icons

Array representation of a single icon

1	1	1	1	3	0	0	0	0	0	0	0
0	0	0	0	3	0	0	0	0	0	0	0
0	0	0	1	3	0	0	0	0	0	0	0
0	0	0	1	3	0	0	0	0	0	0	0
0	0	0	1	2	2	2	2	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	...				

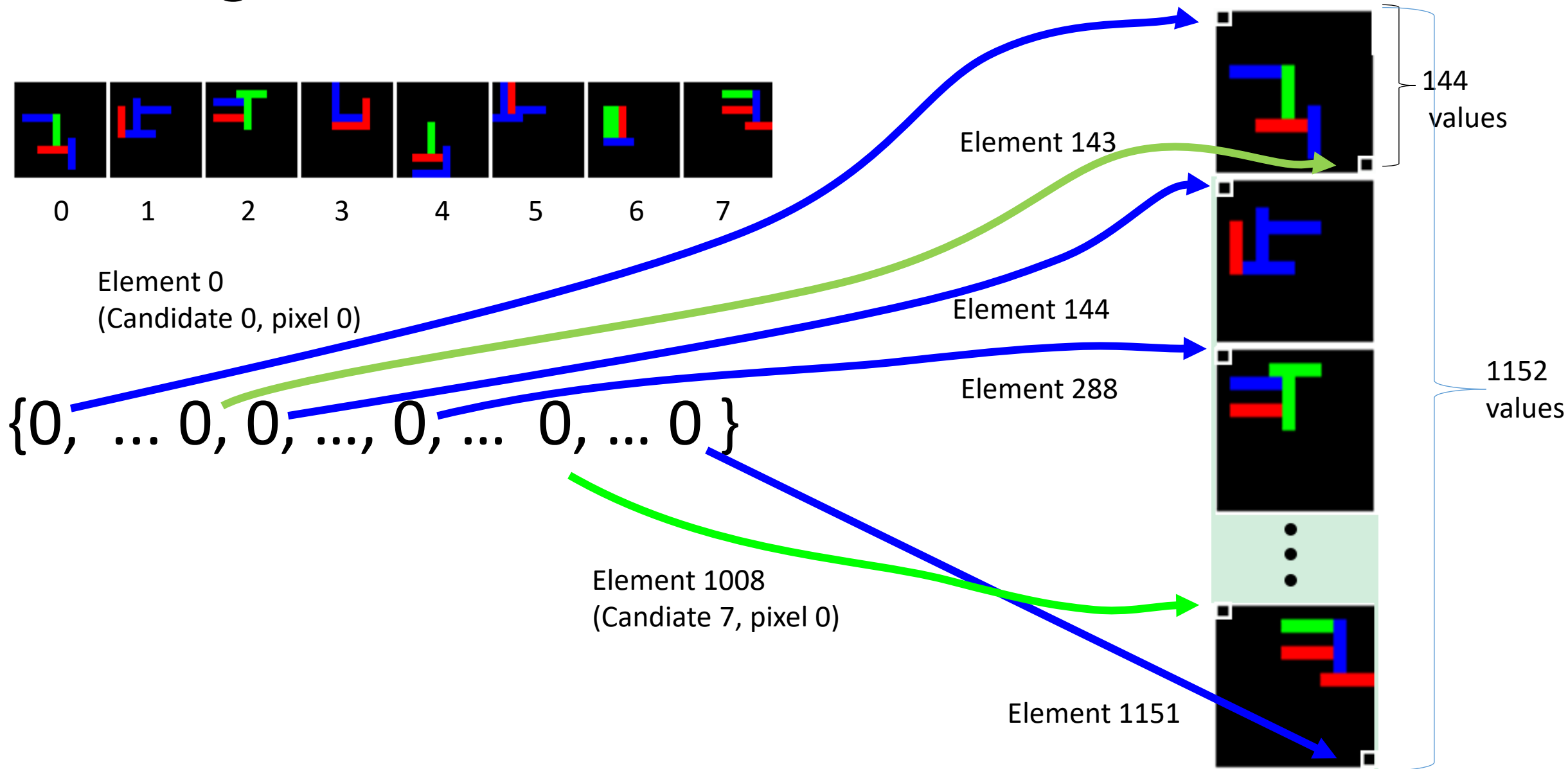
- Row-major
- 1 pixel value per array element
- This example is
{1, 1, 1, 1, 3, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 3, 0, ... }

Array representation of 8 candidate icons



- Each candidate is row-major, exactly as shown on previous slide
- The eight candidates are one after another in a large array
- The image above is misleading – it is more like a 96 row \times 12 column array

Storage order of 8 candidate icons



P1-1: C Implementation

- Declarations:

```
#define IconSize (12 * 12)  ← IconSize is the constant 144
#define NumCandidates 8
#define CandidateArraySize (IconSize * NumCandidates)  ← 144 × 8 = 1152
int Candidates[CandidateArraySize];
int Pattern[IconSize]
```

Stores 1152 ints for Candidates, followed by 144 for Pattern
(just as MIPS version will do)
It COULD HAVE been 1152 bytes, but they are ints!

- Grading:

// your program should use this print statement to report the answer

```
printf("The matching icon is %d\n", Match);
```

Shell code sets Match=42, so change that!

- Functional version: not evaluated on performance, only accuracy

P1-2: MIPS Implementation

`.data`

`CandBase: .alloc 1152`

`PatternBase: .alloc 144`

`.text`

`...`

`# followed by SWI calls and your solution`

- Space for $8 \times 12 \times 12$ candidates
- Space for $1 \times 12 \times 12$ Pattern
- Same order as in C version

P1-2: Software Interrupts

1. SWI 584: Create Puzzle

INPUTS: \$1 = base address

OUTPUTS: none - memory populated with image array

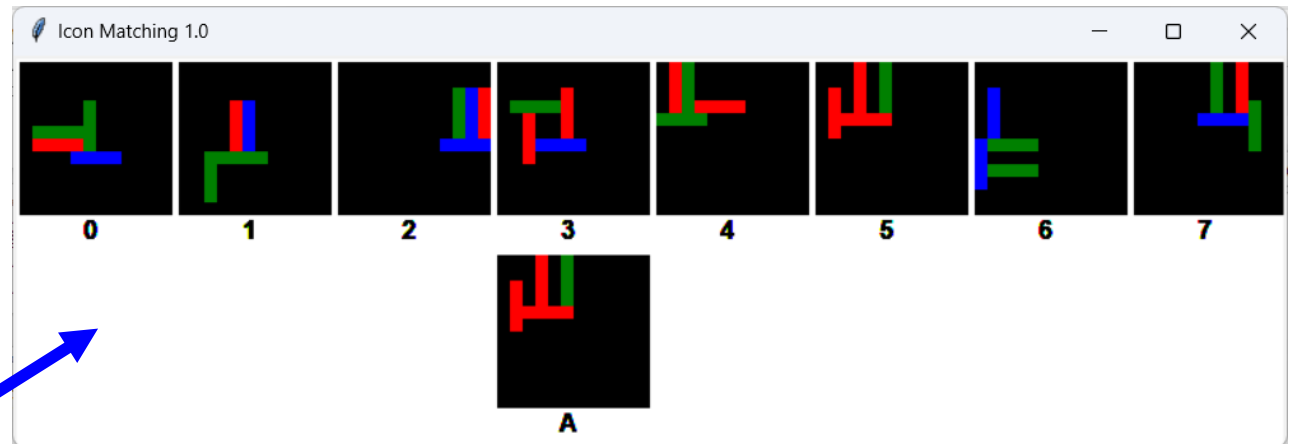
.data

CandBase: .alloc 1152

PatternBase: .alloc 144

.text

```
IconMatch:  addi    $1, $0, CandBase    # point to base of Candidates
              swi     584                # generate puzzle icons
```

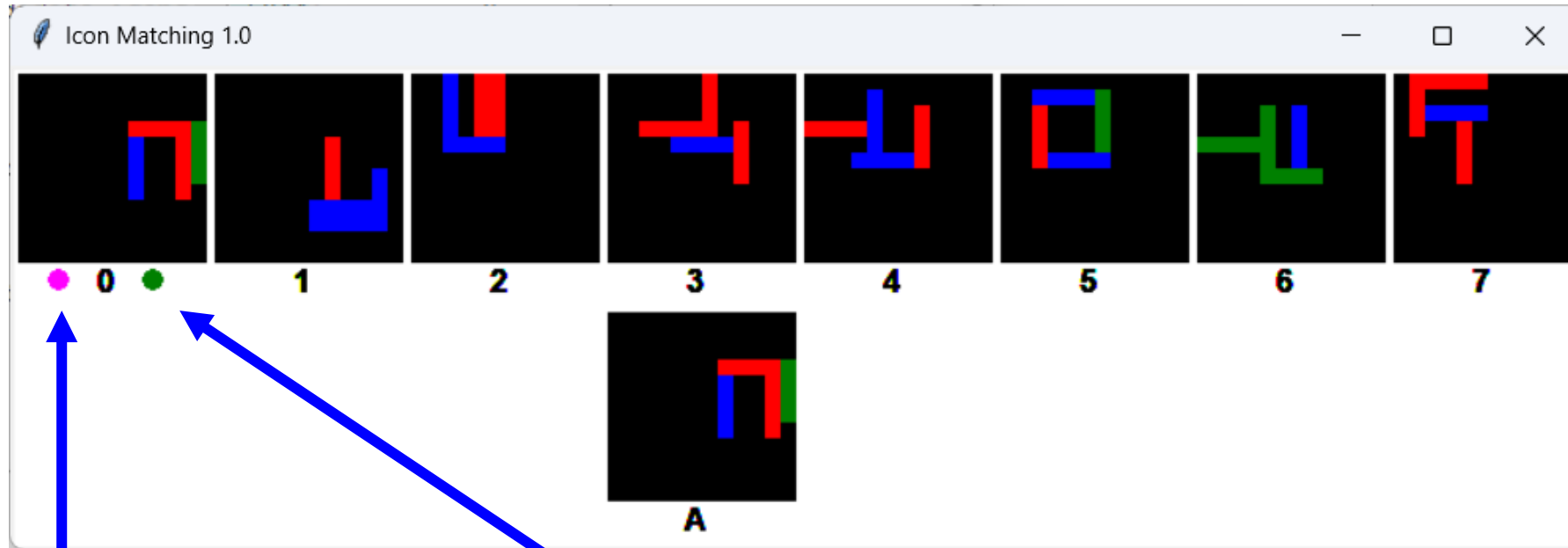


P1-2: Software Interrupts

2. SWI 544: Match Ref: This routine allows you to report the number of the candidate icon that matches the pattern icon.

INPUTS: \$2 a number between 0 and 7, inclusive (candidate #)

OUTPUTS: \$3 gives the correct answer from oracle



submitted answer (magenta dot)

oracle's correct answer (green dot)

P1-2: Software Interrupts

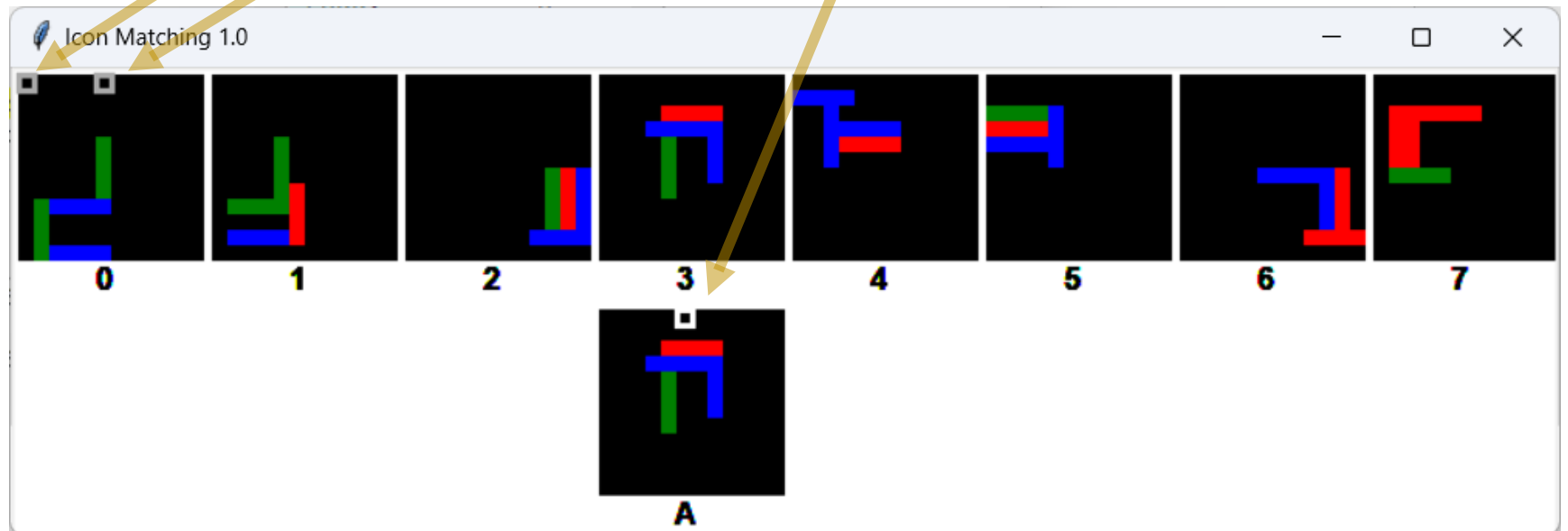
3. SWI 585: Mark Icon Pixel:

INPUTS: \$2 should contain an address of a pixel. The address should be within the 1296 word region allocated for Candidates and the Pattern

Cells that have been highlighted previously in the trace are drawn with a gray outline.

OUTPUTS: none.

```
addi    $2, $0, CandBase # 0th icon, 0th cell
swi     585
addi    $2, $2, 20        # 0th icon, cell 5
swi     585
addi    $2, $0, PatternBase
addi    $2, $2, 20        # pattern, cell 5
swi     585
```

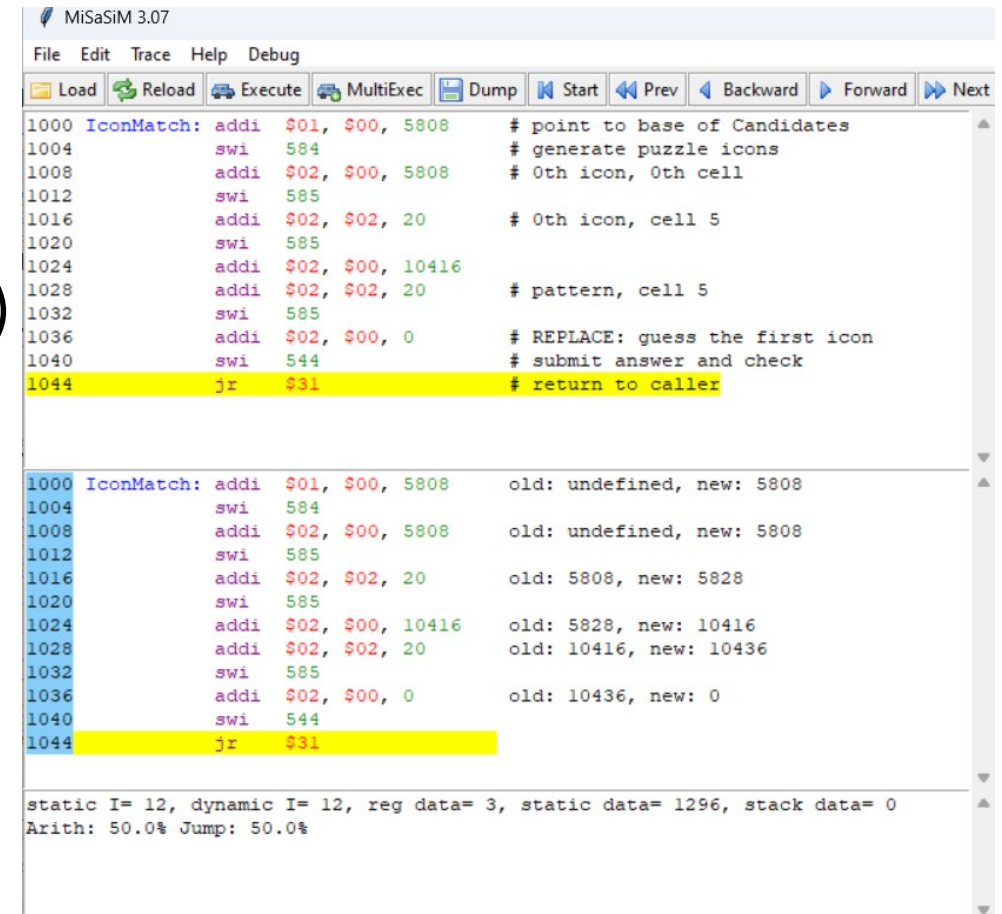


For debugging only

P1-2: Accuracy *and* Efficiency Evaluated

static I= 12, dynamic I= 12, reg data= 3, static data= 1296, stack data= 0
Arith: 50.0% Jump: 50.0%

- Statistics for shell code above
- Very efficient (but tests show not very accurate)



The screenshot shows the MiSaSiM 3.07 application window. The title bar is 'MiSaSiM 3.07'. The menu bar includes 'File', 'Edit', 'Trace', 'Help', and 'Debug'. The toolbar contains icons for 'Load', 'Reload', 'Execute', 'MultiExec', 'Dump', 'Start', 'Prev', 'Backward', 'Forward', and 'Next'. The main window displays assembly code for a function named 'IconMatch'. The code is as follows:

```
1000 IconMatch: addi $01, $00, 5808    # point to base of Candidates
1004           swi  584                # generate puzzle icons
1008           addi $02, $00, 5808    # 0th icon, 0th cell
1012           swi  585
1016           addi $02, $02, 20      # 0th icon, cell 5
1020           swi  585
1024           addi $02, $00, 10416   # pattern, cell 5
1028           addi $02, $02, 20
1032           swi  585
1036           addi $02, $00, 0       # REPLACE: guess the first icon
1040           swi  544               # submit answer and check
1044           jr   $31              # return to caller
```

Below the assembly code, there is a table showing the state of registers and memory locations. The table has two columns: 'old' and 'new'. The rows are as follows:

Address	Instruction	Old	New
1000	IconMatch: addi \$01, \$00, 5808	undefined	5808
1004	swi 584	undefined	5808
1008	addi \$02, \$00, 5808	undefined	5808
1012	swi 585	5808	5828
1016	addi \$02, \$02, 20	5828	10416
1020	swi 585	10416	10436
1024	addi \$02, \$00, 10416	10436	0
1028	addi \$02, \$02, 20	0	0
1032	swi 585	0	0
1036	addi \$02, \$00, 0	0	0
1040	swi 544	0	0
1044	jr \$31	0	0

At the bottom of the window, the execution statistics are displayed:

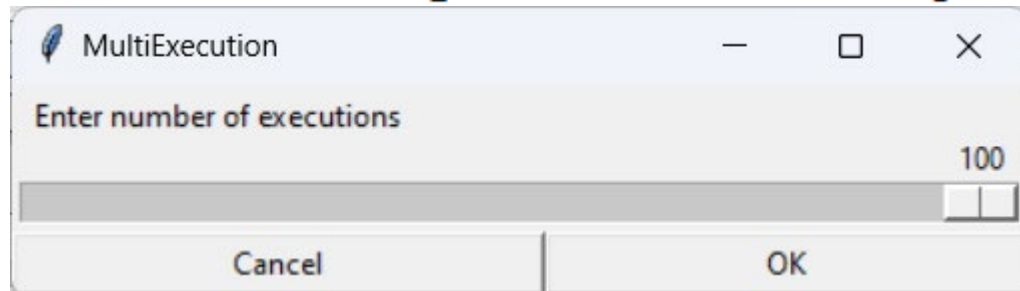
```
static I= 12, dynamic I= 12, reg data= 3, static data= 1296, stack data= 0
Arith: 50.0% Jump: 50.0%
```

P1-2: Efficiency Evaluation

- Your Goal: Beat the baseline
static code size: 38 instructions, dynamic instruction length: 350 instructions (avg.), total register and memory storage: 11 words (not including dedicated registers \$0, \$31, or the 1296 words for the input puzzle array)
- You should run MultiExecute to get average DI, storage over multiple runs

```
static I= 30, dynamic I= 327, reg data= 7, static data= 1024, stack data= 0  
Arith: 34.3% Jump: 0.9% Load: 32.4% Branch: 32.4%
```

```
static I= 30, dynamic I= 1179, reg data= 7, static data= 1024, stack data= 0  
Arith: 34.3% Jump: 0.9% Load: 32.4% Branch: 33.1%
```



We'll run 100 trials (same trials for all students).

Score

		Part	Description	Points
Accuracy Metrics	P1-1		Icon Matching (C code)	25
	P1-2		Icon Matching (MIPS code)	
Performance Metrics			checkpoints, correct operation, proper commenting/style	25
			static code size	15
			dynamic execution length	25
			storage requirements (registers, memory)	10
			Total	100

For Accuracy Metrics:

- Points reduced by 10% for each failed trial
- 0 points if 10 or more trials fail AND (for P1-2) no points for any performance metrics

Score

		Part	Description	Points
Accuracy Metrics	P1-1		Icon Matching (C code)	25
	P1-2		Icon Matching (MIPS code)	
Performance Metrics			checkpoints, correct operation, proper commenting/style	25
			static code size	15
			dynamic execution length	25
			storage requirements (registers, memory)	10
			Total	100

For each P1-2 Performance Metric:

Points = PercentCredit x (Performance Metric Points)

where

$$\text{PercentCredit} = 2 - \frac{\text{Metric}_{\text{Your Program}}}{\text{Metric}_{\text{Baseline Program}}}$$

Baseline Metrics:

static code size: 38 instructions,

dynamic inst length: 350 instructions (avg.),

total register & memory storage: 11 words

E.g., if your program has a 315 avg dynamic execution length,

Points for dynamic: $(2 - 315/350) \times 25 = 1.1 \times 25 = 27.5$ (out of 25)

If your program uses 12 words, storage points: $(2 - 12/10) \times 10 = 8$ (out of 10)

P1-2 swi 584 Easter Egg

```
addi $1, $0, CandBase  
addi $2, $0, -1  
swi 584
```

- Ability to load in a previously dumped puzzle
- Debugging feature
- **Be sure to comment out this change to \$2 before submitting your code** so that the autograder is not prompted for a test file.

Honor Code

In all programming assignments, you should design, implement, and test your own code. Any submitted assignment containing non-shell code that is not fully created and debugged by the student constitutes academic misconduct. You should not share code, debug code, or discuss its performance with anyone. *Once you begin implementing your solution, you must work alone.*

Copyright 2024 Georgia Tech. All rights reserved. The materials provided by the instructor in this course are for the use of the students currently enrolled in the course. Copyrighted course materials may not be further disseminated. This file must not be made publicly available anywhere.