*ARKO 21L*

***Projekt***

**MIPS**

*Richard Staszkiewicz*

[Zadanie 3](#_Toc71394521)

[Treść 3](#_Toc71394522)

[Specyfikacja strumieni 3](#_Toc71394523)

[Przyjęte założenia 3](#_Toc71394524)

[Analiza problemu 4](#_Toc71394525)

[Algorytm rozwiązania 5](#_Toc71394526)

[Przygotowanie i odczyt danych 5](#_Toc71394527)

[Rozpoznanie środowiska 5](#_Toc71394528)

[Testy 6](#_Toc71394529)

[Wyświetlenie wyniku 6](#_Toc71394530)

[Funkcje i Rejestry 7](#_Toc71394531)

[Podstawowe funkcje i ich działanie. 7](#_Toc71394532)

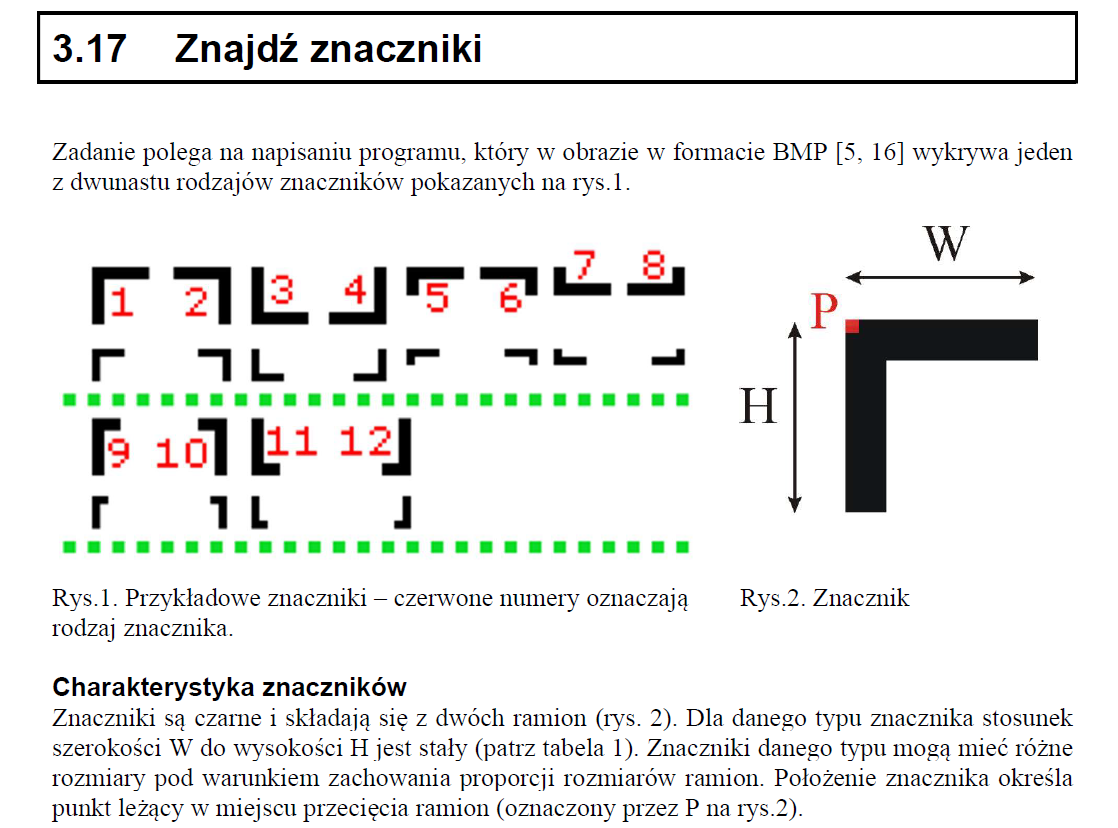
[Rejestry funkcji find\_marker 7](#_Toc71394533)

[Załączniki 7](#_Toc71394534)

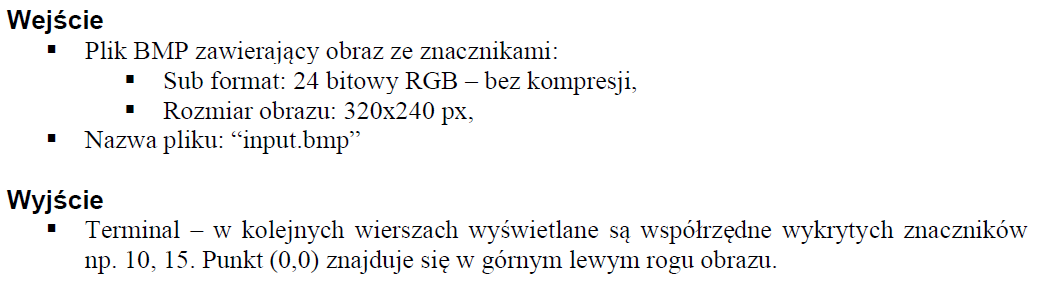
# Zadanie

Przydzielonym zadaniem jest zadanie 3.17: Znajdź znaczniki (znacznik nr. 4) z pliku Materiały pomocnicze do laboratorium aut. mgr. inż. Zbigniewa Szymańskiego z marca b. r.

## Treść



## Specyfikacja strumieni



## Przyjęte założenia

W celu zachowania dodatnich współrzędnych punkt (0,0) znajduje się w lewym dolnym rogu obrazka.

# Analiza problemu

Problem zasadniczo sprowadza się do wyodrębnienia określonego, monochromatycznego kształtu na mapie bitowej. W zadanym wypadku jest nieznacznie prościej, gdyż z założeń zadania wiadomo że jedyne rozpatrywane kształty mają kolor **czarny**.

Aby wyodrębnić kształt, należy zidentyfikować wszystkie piksele jednej barwy, przylegające do siebie bokami (przyległych rogami nie liczymy jako sąsiednich). Po zidentyfikowaniu ich wszystkich, należy sprawdzić czy zachowują one kształt opisany w zadaniu i wypisać koordynaty ***punktu charakterystycznego***.

Ze względu na charakter problemu, każdy z pikseli zadanej bitmapy należy sprawdzić przynajmniej raz, a więc minimalna złożoność problemu wynosi **O(*nm*)** gdzie *n* i *m* są wymiarami bitmapy (w tej sytuacji stałymi równymi odpowiednio **240** i **320**. Poniższe rozwiązanie działa w złożoności **O(*nm*)**, każdy piksel dotykając w najgorszym wypadku 2 razy.

Z optymalizacyjnego punktu widzenia, należy mieć na względzie również ograniczenie ilości **dostępów do pamięci** oraz redukcję **skoków**.

Ze względu na założenia zadania, **nie są** **uznawane** za markery kształty składające się z nich jedynie w części. Ze względu na założenia zadania **uznano** również kwadraty jako markery, aczkolwiek pojedyncze piksele są **odrzucane** jako szum.

# Algorytm rozwiązania

## Przygotowanie i odczyt danych

W sekcji .data zostaje zarezerwowane miejsce na obrabianą bitmapę oraz ok. trzykrotnie mniejsze miejsce, które posłuży do przechowywania mapy zużytych pikseli. ***Zużytym pikselem*** jest piksel o jednoznacznie rozstrzygniętym stanie (bądź należenia do markera, bądź nienależenia). Alignment zostaje ustawiony standardowo, na 4 bajty (można by próbować optymalizacji, ale byłaby to raczej niewielka zmiana).

Za pomocą funkcji **read\_bmp** następuje odczyt danych z zadanego w kodzie pliku źródłowego. W wypadku próby odczytania już otwartego pliku bądź błędów przy wczytywaniu, wywoływana jest informacja o błędzie i program kończy działanie. W przeciwnym wypadku, plik zostaje załadowany do zarezerwowanej pamięci programu.

Za pomocą funkcji **clear\_used**, zarezerwowane przez program miejsce na mapę zużytych pikseli zostaje wyzerowane. Jest to jedynie środek ostrożności, w wypadku gdyby program przez przypadek zarezerwował miejsce z jakimiś pozostałościami dawnych danych.

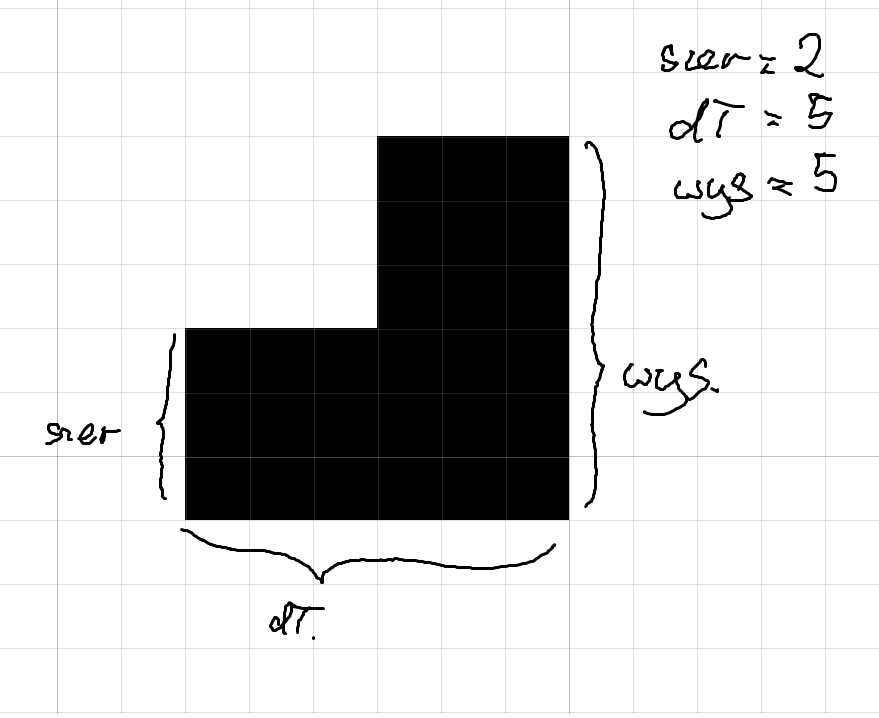
Następnie dla każdego piksela zostaje wywołana funkcja **find\_marker**.

## Rozpoznanie środowiska

Obliczany jest adres piksela w badanej bitmapie oraz w tablicy zużytych (oznaczając go przy okazji jako zużyty). Za pomocą funkcji **get\_pixel**, badane jest RGB rozpatrywanego piksela – w dalszych krokach są rozważane jedynie czarne.

Wówczas flaga **print** przechowywana pod adresem $s5 zostaje ustawiona na *True*. Pod koniec czynności weryfikacyjnych, to od niej zależeć będzie, czy marker został znaleziony poprawny i należy go wypisać, czy też kształt nie spełnia wymogów zadania.

Następnie obliczane są trzy potencjalne wymiary markera, odpowiednio długość, szerokość i wysokość, obliczane z pomocą funkcji **get\_len** i **get\_hgh**. Obie funkcje zwracają odpowiednio liczbę czarnych pikseli ponad i po prawej zadanego, zaznaczając je przy okazji jako zużyte. Odrębną zmienną sygnalizują błąd, jeżeli którykolwiek z tych pikseli był już zużyty. Obrazek poniżej pokazuje ich wyniki dla przykładowego kształtu.



## Testy

Weryfikacja składa się z 7 testów, wykonywanych niezależnie od siebie.

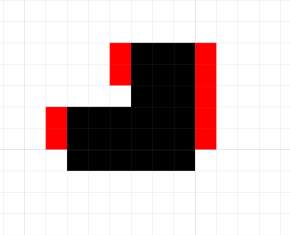
**Test 1**: sprawdza czy wysokość potencjalnego markera jest równa jego długości.

**Test 2**: sprawdza czy szerokość pionowego ramienia potencjalnego markera jest taka sama jak poziomego.

**Test 3**: sprawdza czy ramię pionowe jest kompletne, tj. nie ma brakujących pikseli w prostokącie.

**Test 4**: sprawdza czy ramię poziome jest kompletne, tj. nie ma brakujących pikseli w prostokącie.

**Testy 5, 6, 7**: Weryfikuje czy w okolicy niesprawdzonych granic potencjalnego markera nie ma odrostów. Ilustracja badanych pikseli krawędzi poniżej. Sprawdzanie najbardziej wysuniętych krańców jest zaniechiwane w wypadku przyległości potencjalnego markera do ścian planszy.



## Wyświetlenie wyniku

Jeżeli po testach flaga **print** z rejestru $s5 jest w dalszym ciągu ustawiona na *True*, następuje wypisanie na wyjście standardowe przesunięte o odpowiednią wartość współrzędne obecnie rozpatrywanego piksela.

# Funkcje i Rejestry

## Podstawowe funkcje i ich działanie.

|  |  |  |  |
| --- | --- | --- | --- |
| **Funkcja główna** | **Funkcja pomocnicza** | **Argumenty** | **Wynik** |
| Read\_bmp | - | - | *Wczytanie bitmapy spod zadanej ścieżki, bądź wywołanie informacji o błędzie* |
| Clear\_used | - | - | *Wyzerowanie miejsca na mapę zużytych pikseli* |
| Find\_marker | - | **$a0** – współrzędna X **$a1** – współrzędna Y | *Jeżeli piksel znajduje się w lewym dolnym rogu markera, wypisywane są koordynaty markera* |
| => | Get\_pixel | **$a0** – współrzędna X **$a1** – współrzędna Y  **$a2** – Adres piksela w bitmapie | **$v0** – 0RGB, kolor piksela  *Funkcja pobiera kolor piksela* |
| => | Get\_len | **$a0** – współrzędna X **$a1** – współrzędna Y  **$a2** – Adres piksela w bitmapie  **$a3** – Adres piksela w użytych | **$v0** – delta długości  **$v1** – Czy poprawne? *Funkcja oznacza przejrzane piksele jako zużyte* |
| => | Get\_hgh | **$a0** – współrzędna X **$a1** – współrzędna Y  **$a2** – Adres piksela w bitmapie  **$a3** – Adres piksela w użytych | **$v0** – delta długości  **$v1** – Czy poprawne?  *Funkcja oznacza przejrzane piksele jako zużyte* |
| => | Edge\_v | **$a0** – współrzędna X **$a1** – współrzędna Y  **$a2** – Adres piksela w bitmapie  **$a3** – Adres piksela w użytych  **$s7** – Ilość px do sprawdzenia | **$v1** – Czy poprawna?  *Funkcja sprawdza czy pionowa krawędź markera jest poprawna* |

## Rejestry funkcji find\_marker

|  |  |
| --- | --- |
| **Rejestr** | **Przechowywana wartość** |
| $a0 | Współrzędna **X** rozpatrywanego piksela |
| $a1 | Współrzędna **Y** rozpatrywanego piksela |
| $s0 | Adres w **zużytych** rozpatrywanego piksela |
| $s1 | Adres w **bitmapie** rozpatrywanego piksela |
| $s2 | Potencjalna **długość** rozpatrywanego markera |
| $s3 | Potencjalna **szerokość** rozpatrywanego markera |
| $s4 | Potencjalna **wysokość** rozpatrywanego markera |
| $s5 | Flaga **print** |

# Załączniki

1. Plik MIPS.asm z realizacją komputerową zadania (wraz z komentarzami)
2. Plik testowy